HMAC authentication for the Babel routing protocol
draft-ietf-babel-hmac-04

Abstract

This document describes a cryptographic authentication mechanism for
the Babel routing protocol that has provisions for replay avoidance.
This document updates RFC 6126bis and obsoletes RFC 7298.

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 https://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 10, 2019.

Copyright Notice

Copyright (c) 2019 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
(https://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
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
1. Introduction

By default, the Babel routing protocol trusts the information contained in every UDP packet it receives on the Babel port. An attacker can redirect traffic to itself or to a different node in the network, causing a variety of potential issues. In particular, an attacker might:

- spoof a Babel packet, and redirect traffic by announcing a smaller metric, a larger seqno, or a longer prefix;

- spoof a malformed packet, which could cause an insufficiently robust implementation to crash or interfere with the rest of the network;
replay a previously captured Babel packet, which could cause traffic to be redirected or otherwise interfere with the network.

Protecting a Babel network is challenging due to the fact that the Babel protocol uses both unicast and multicast communication. One possible approach, used notably by the Babel over Datagram Transport Layer Security (DTLS) protocol [I-D.ietf-babel-dtls], is to use unicast communication for all semantically significant communication, and then use a standard unicast security protocol to protect the Babel traffic. In this document, we take the opposite approach: we define a cryptographic extension to the Babel protocol that is able to protect both unicast and multicast traffic, and thus requires very few changes to the core protocol.

1.1. Applicability

The protocol defined in this document assumes that all interfaces on a given link are equally trusted and share a small set of symmetric keys (usually just one, and two during key rotation). The protocol is inapplicable in situations where asymmetric keying is required, where the trust relationship is partial, or where large numbers of trusted keys are provisioned on a single link at the same time.

This protocol supports incremental deployment (where an insecure Babel network is made secure with no service interruption), and it supports graceful key rotation (where the set of keys is changed with no service interruption).

This protocol does not require synchronised clocks, it does not require persistently monotonic clocks, and it does not require persistent storage except for what might be required for storing cryptographic keys.

1.2. Assumptions and security properties

The correctness of the protocol relies on the following assumptions:

- that the Hashed Message Authentication Code (HMAC) being used is invulnerable to pre-image attacks, i.e., that an attacker is unable to generate a packet with a correct HMAC;

- that a node never generates the same index or nonce twice over the lifetime of a key.

The first assumption is a property of the HMAC being used. The second assumption can be met either by using a robust random number generator [RFC4086] and sufficiently large indices and nonces, by
using a reliable hardware clock, or by rekeying whenever a collision becomes likely.

If the assumptions above are met, the protocol described in this document has the following properties:

- it is invulnerable to spoofing: any packet accepted as authentic is the exact copy of a packet originally sent by an authorised node;
- locally to a single node, it is invulnerable to replay: if a node has previously accepted a given packet, then it will never again accept a copy of this packet or an earlier packet from the same sender;
- among different nodes, it is only vulnerable to immediate replay: if a node A has accepted a packet from C as valid, then a node B will only accept a copy of that packet as authentic if B has accepted an older packet from C and B has received no later packet from C.

While this protocol makes serious efforts to mitigate the effects of a denial of service attack, it does not fully protect against such attacks.

1.3. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Conceptual overview of the protocol

When a node B sends out a Babel packet through an interface that is configured for HMAC cryptographic protection, it computes one or more HMACs which it appends to the packet. When a node A receives a packet over an interface that requires HMAC cryptographic protection, it independently computes a set of HMACs and compares them to the HMACs appended to the packet; if there is no match, the packet is discarded.

In order to protect against replay B maintains a per-interface 32-bit integer known as the "packet counter" (PC). Whenever B sends a packet through the interface, it embeds the current value of the PC within the region of the packet that is protected by the HMACs and increases the PC by at least one. When A receives the packet, it
compares the value of the PC with the one contained in the previous packet received from B, and unless it is strictly greater, the packet is discarded.

By itself, the PC mechanism is not sufficient to protect against replay. Consider a peer A that has no information about a peer B (e.g., because it has recently rebooted). Suppose that A receives a packet ostensibly from B carrying a given PC; since A has no information about B, it has no way to determine whether the packet is freshly generated or a replay of a previously sent packet.

In this situation, A discards the packet and challenges B to prove that it knows the HMAC key. It sends a "challenge request", a TLV containing a unique nonce, a value that has never been used before and will never be used again. B replies to the challenge request with a "challenge reply", a TLV containing a copy of the nonce chosen by A, in a packet protected by HMAC and containing the new value of B’s PC. Since the nonce has never been used before, B’s reply proves B’s knowledge of the HMAC key and the freshness of the PC.

By itself, this mechanism is safe against replay if B never resets its PC. In practice, however, this is difficult to ensure, as persistent storage is prone to failure, and hardware clocks, even when available, are occasionally reset. Suppose that B resets its PC to an earlier value, and sends a packet with a previously used PC n. A challenges B, B successfully responds to the challenge, and A accepts the PC equal to n + 1. At this point, an attacker C may send a replayed packet with PC equal to n + 2, which will be accepted by A.

Another mechanism is needed to protect against this attack. In this protocol, every PC is tagged with an "index", an arbitrary string of octets. Whenever B resets its PC, or whenever B doesn’t know whether its PC has been reset, it picks an index that it has never used before (either by drawing it randomly or by using a reliable hardware clock) and starts sending PCs with that index. Whenever A detects that B has changed its index, it challenges B again.

With this additional mechanism, this protocol is invulnerable to replay attacks (see Section 1.2 above).

3. Data Structures

Every Babel node maintains a set of conceptual data structures described in [RFC6126bis] Section 3.2. This protocol extends these data structures as follows.
3.1. The Interface Table

Every Babel node maintains an interface table, as described in [RFC6126bis] Section 3.2.3. Implementations of this protocol MUST allow each interface to be provisioned with a set of one or more HMAC keys and the associated HMAC algorithms (see Section 4.1). In order to allow incremental deployment of this protocol (see Appendix A), implementations SHOULD allow an interface to be configured in a mode in which it participates in the HMAC authentication protocol but accepts packets that are not authenticated.

This protocol extends each entry in this table that is associated with an interface on which HMAC authentication has been configured with two new pieces of data:

- a set of one or more HMAC keys, each associated with a given HMAC algorithm; the length of each key is exactly the hash size of the associated HMAC algorithm (i.e., the key is not subject to the preprocessing described in Section 2 of [RFC2104]);

- a pair (Index, PC), where Index is an arbitrary string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer.

The Index and PC are initialised to arbitrary values chosen so as to ensure that a given (Index, PC) pair is never reused. Typically, the initial Index will be chosen as a random string of sufficient length, and the initial PC will be set to 0.

3.2. The Neighbour Table

Every Babel node maintains a neighbour table, as described in [RFC6126bis] Section 3.2.4. This protocol extends each entry in this table with two new pieces of data:

- a pair (Index, PC), where Index is a string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer;

- a Nonce, an arbitrary string of 0 to 192 octets, and an associated challenge expiry timer.

The Index and PC are initially undefined, and are managed as described in Section 4.3. The Nonce and expiry timer are initially undefined, and used as described in Section 4.3.1.1.
4. Protocol Operation

4.1. HMAC computation

A Babel node computes an HMAC as follows.

First, the node builds a pseudo-header that will participate in HMAC computation but will not be sent. If the packet was carried over IPv6, the pseudo-header has the following format:

```
+----------------------------------+
|       Src address                |
+----------------------------------+
|       Src port                   |
|                             +----+
| Dest address                   |
+----------------------------------+

Fields:
- Src address: The source IP address of the packet.
```

If the packet was carried over IPv4, the pseudo-header has the following format:

```
+----------------------------------+
|       Src address                |
|                             +----+
| Dest address                   |
|                             +----+
|       Dest port                 |
+----------------------------------+

Fields:
- Src address: The source IP address of the packet.
The node takes the concatenation of the pseudo-header and the packet including the packet header but excluding the packet trailer (from octet 0 inclusive up to Body Length + 4 exclusive) and computes an HMAC with one of the implemented hash algorithms. Every implementation MUST implement HMAC-SHA256 as defined in [RFC6234] and Section 2 of [RFC2104], SHOULD implement keyed BLAKE2s [RFC7693], and MAY implement other HMAC algorithms.

4.2. Packet Transmission

A Babel node may delay actually sending TLVs by a small amount, in order to aggregate multiple TLVs in a single packet up to the interface MTU (Section 4 of [RFC6126bis]). For an interface on which HMAC protection is configured, the TLV aggregation logic MUST take into account the overhead due to PC TLVs (one in each packet) and HMAC TLVs (one per configured key).

Before sending a packet, the following actions are performed:

- a PC TLV containing the PC and Index associated with the outgoing interface is appended to the packet body; the PC is incremented by a strictly positive amount (typically just 1); if the PC overflows, a fresh index is generated;

- for each key configured on the interface, an HMAC is computed as specified in Section 4.1 above, and an HMAC TLV is appended to the packet trailer (see Section 4.2 of [RFC6126bis]).

4.3. Packet Reception

When a packet is received on an interface that is configured for HMAC protection, the following steps are performed before the packet is passed to normal processing:

- First, the receiver checks whether the trailer of the received packet carries at least one HMAC TLV; if not, the packet is immediately dropped and processing stops. Then, for each key configured on the receiving interface, the implementation computes the HMAC of the packet. It then compares every generated HMAC against every HMAC included in the packet; if there is at least one match, the packet passes the HMAC test; if there is none, the packet is silently dropped and processing stops at this point. In
order to avoid memory exhaustion attacks, an entry in the Neighbour Table MUST NOT be created before the HMAC test has passed successfully. The HMAC of the packet MUST NOT be computed for each HMAC TLV contained in the packet, but only once for each configured key.

- The packet body is then parsed a first time. During this "preparse" phase, the packet body is traversed and all TLVs are ignored except PC TLVs, Challenge Requests and Challenge Replies. When a PC TLV is encountered, the enclosed PC and Index are saved for later processing; if multiple PCs are found, only the first one is processed, the remaining ones are silently ignored. If a Challenge Request is encountered, a Challenge Reply is scheduled, as described in Section 4.3.1.2, and if a Challenge Reply is encountered, it is tested for validity as described in Section 4.3.1.3 and a note is made of the result of the test.

- The preparse phase above has yielded two pieces of data: the PC and Index from the first PC TLV, and a bit indicating whether the packet contains a successful Challenge Reply. If the packet does not contain a PC TLV, the packet is dropped and processing stops at this point. If the packet contains a successful Challenge Reply, then the PC and Index contained in the PC TLV are stored in the Neighbour Table entry corresponding to the sender (which may need to be created at this stage), and the packet is accepted.

- Otherwise, if there is no entry in the Neighbour Table corresponding to the sender, or if such an entry exists but contains no Index, or if the Index it contains is different from the Index contained in the PC TLV, then a challenge is sent as described in Section 4.3.1.1, processing stops at this stage, and the packet is dropped.

- At this stage, the packet contained no successful challenge reply and the Index contained in the PC TLV is equal to the Index in the Neighbour Table entry corresponding to the sender. The receiver compares the received PC with the PC contained in the Neighbour Table; if the received PC is smaller or equal than the PC contained in the Neighbour Table, the packet is silently dropped and processing stops (no challenge is sent in this case, since the mismatch might be caused by harmless packet reordering on the link). Otherwise, the PC contained in the Neighbour Table entry is set to the received PC, and the packet is accepted.

After the packet has been accepted, it is processed as normal, except that any PC, Challenge Request and Challenge Reply TLVs that it contains are silently ignored.
4.3.1. Challenge Requests and Replies

During the preparse stage, the receiver might encounter a mismatched Index, to which it will react by scheduling a Challenge Request. It might encounter a Challenge Request TLV, to which it will reply with a Challenge Reply TLV. Finally, it might encounter a Challenge Reply TLV, which it will attempt to match with a previously sent Challenge Request TLV in order to update the Neighbour Table entry corresponding to the sender of the packet.

4.3.1.1. Sending challenges

When it encounters a mismatched Index during the preparse phase, a node picks a nonce that it has never used before, for example by drawing a sufficiently large random string of bytes or by consulting a strictly monotonic hardware clock. It stores the nonce in the entry of the Neighbour Table of the neighbour (the entry might need to be created at this stage), initialises the neighbour’s challenge expiry timer to 30 seconds, and sends a Challenge Request TLV to the unicast address corresponding to the neighbour.

A node MAY aggregate a Challenge Request with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the sender of the neighbour, it MAY send the buffered TLVs in the same packet as the Challenge Request. However, it MUST arrange for the Challenge Request to be sent in a timely manner, as any packets received from that neighbour will be silently ignored until the challenge completes.

Since a challenge may be prompted by a replayed packet, a node MUST impose a rate limitation to the challenges it sends; the limit SHOULD default to one challenge request every 300ms, and MAY be configurable.

4.3.1.2. Replying to challenges

When it encounters a Challenge Request during the preparse phase, a node constructs a Challenge Reply TLV by copying the Nonce from the Challenge Request into the Challenge Reply. It sends the Challenge Reply to the unicast address from which the Challenge Request was sent.

A node MAY aggregate a Challenge Reply with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the sender of the Challenge Request, it MAY send the buffered TLVs in the same packet as the Challenge Reply. However, it MUST arrange for the Challenge Reply to be sent in a timely manner (within a few seconds), and SHOULD NOT send any other packets over
the same interface before sending the Challenge Reply, as those would
be dropped by the challenger.

A challenge sent to a multicast address MUST be silently ignored.

4.3.1.3. Receiving challenge replies

When it encounters a Challenge Reply during the preparse phase, a
node consults the Neighbour Table entry corresponding to the
neighbour that sent the Challenge Reply. If no challenge is in
progress, i.e., if there is no Nonce stored in the Neighbour
Table entry or the Challenge timer has expired, the Challenge Reply
is silently ignored and the challenge has failed.

Otherwise, the node compares the Nonce contained in the Challenge
Reply with the Nonce contained in the Neighbour Table entry. If the
two are equal (they have the same length and content), then the
challenge has succeeded; otherwise, the challenge has failed.

4.4. Expiring per-neighbour state

The per-neighbour (Index, PC) pair is maintained in the neighbour
table, and is normally discarded when the neighbour table entry
expires. Implementations MUST ensure that an (Index, PC) pair is
discarded within a finite time since the last time a packet has been
accepted. In particular, unsuccessful challenges MUST NOT prevent an
(Index, PC) pair from being discarded for unbounded periods of time.

A possible implementation strategy for implementations that use a
Hello history (Appendix A of [RFC6126bis]) is to discard the (Index,
PC) pair whenever the Hello history becomes empty. Another
implementation strategy is to use a timer that is reset whenever a
packet is accepted, and to discard the (Index, PC) pair whenever the
timer expires. If the latter strategy is being used, the timer
SHOULD default to a value of 5 min, and MAY be configurable.

5. Packet Format

5.1. HMAC TLV

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 16   |    Length     |     HMAC...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Type          Set to 16 to indicate an HMAC TLV.
Length        The length of the body, exclusive of the Type and Length fields. The length of the body depends on the hash function used.
HMAC          The body contains the HMAC of the whole packet plus the pseudo header.

This TLV is allowed in the packet trailer (see Section 4.2 of [RFC6126bis]), and MUST be ignored if it is found in the packet body.

5.2. PC TLV

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 17   |    Length     |             PC
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Index...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:
Type          Set to 17 to indicate a PC TLV.
Length        The length of the body, exclusive of the Type and Length fields.

PC            The Packet Counter (PC), a 32-bit (4 octet) unsigned integer which is increased with every packet sent over this interface. A fresh index MUST be generated whenever the PC overflows.

Index         The sender’s Index, an opaque string of 0 to 32 octets.

Indices are limited to a size of 32 octets: a node MUST NOT send a TLV with an index of size strictly larger than 32 octets, and a node MAY silently ignore a PC TLV with an index of size strictly larger than 32 octets.

5.3. Challenge Request TLV

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 18   |    Length     |     Nonce...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Do, et al. Expires September 10, 2019
Fields:

Type      Set to 18 to indicate a Challenge Request TLV.
Length    The length of the body, exclusive of the Type and Length fields.
Nonce     The nonce uniquely identifying the challenge, an opaque string of 0 to 192 octets.

Nonces are limited to a size of 192 octets: a node MUST NOT send a Challenge Request TLV with a nonce of size strictly larger than 192 octets, and a node MAY ignore a nonce that is of size strictly larger than 192 octets.

5.4. Challenge Reply TLV

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 19   |    Length     |     Nonce...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type      Set to 19 to indicate a Challenge Reply TLV.
Length    The length of the body, exclusive of the Type and Length fields. The length of the body is set to the same size as the challenge request TLV length received.
Nonce     A copy of the nonce contained in the corresponding challenge request.

6. Security Considerations

This document defines a mechanism that provides basic security properties for the Babel routing protocol. The scope of this protocol is strictly limited: it only provides authentication (we assume that routing information is not confidential), it only supports symmetric keying, and it only allows for the use of a small number of symmetric keys on every link. Deployments that need more features, e.g., confidentiality or asymmetric keying, should use a more featureful security mechanism such as the one described in [I-D.ietf-babel-dtls].

This mechanism relies on two assumptions, as described in Section 1.2. First, it assumes that the hash being used is
invulnerable to pre-image attacks (Section 1.1 of [RFC6039]); at the time of writing, SHA-256, which is mandatory to implement (Section 4.1), is believed to be safe against practical attacks.

Second, it assumes that indices and nonces are generated uniquely over the lifetime of a key used for HMAC computation (more precisely, indices must be unique for a given (key, source) pair, and nonces must be unique for a given (key, source, destination) triple). This property can be satisfied either by using a cryptographically secure random number generator to generate indices and nonces that contain enough entropy (64-bit values are believed to be large enough for all practical applications), or by using a reliably monotonic hardware clock. If uniqueness cannot be guaranteed (e.g., because a hardware clock has been reset), then rekeying is necessary.

The expiry mechanism mandated in Section 4.4 is required to prevent an attacker from delaying an authentic packet by an unbounded amount of time. If an attacker is able to delay the delivery of a packet (e.g., because it is located at a layer 2 switch), then the packet will be accepted as long as the corresponding (Index, PC) pair is present at the receiver. If the attacker is able to cause the (Index, PC) pair to persist for arbitrary amounts of time (e.g., by causing failed challenges), then it is able to delay the packet by arbitrary amounts of time, even after the sender has left the network.

While it is probably not possible to be immune against denial of service (DoS) attacks in general, this protocol includes a number of mechanisms designed to mitigate such attacks. In particular, reception of a packet with no correct HMAC creates no local state whatsoever (Section 4.3). Reception of a replayed packet with correct hash, on the other hand, causes a challenge to be sent; this is mitigated somewhat by requiring that challenges be rate-limited.

At first sight, sending a challenge requires retaining enough information to validate the challenge reply. However, the nonce included in a challenge request and echoed in the challenge reply can be fairly large (up to 192 octets), which should in principle permit encoding the per-challenge state as a secure "cookie" within the nonce itself.

7. IANA Considerations

IANA has allocated the following values in the Babel TLV Numbers registry:
8. Acknowledgments

The protocol described in this document is based on the original HMAC protocol defined by Denis Ovsienko [RFC7298]. The use of a pseudo-header was suggested by David Schinazi. The use of an index to avoid replay was suggested by Markus Stenberg. The authors are also indebted to Donald Eastlake, Toke Hoiland-Jorgensen, Florian Horn, and Dave Taht.

9. References

9.1. Normative References


9.2. Informational References

[I-D.ietf-babel-dtls]


Appendix A. Incremental deployment and key rotation

This protocol supports incremental deployment (transitioning from an insecure network to a secured network with no service interruption) and key rotation (transitioning from a set of keys to a different set of keys).

In order to perform incremental deployment, the nodes in the network are first configured in a mode where packets are sent with authentication but not checked on reception. Once all the nodes in the network are configured to send authenticated packets, nodes are reconfigured to reject unauthenticated packets.

In order to perform key rotation, the new key is added to all the nodes; once this is done, both the old and the new key are sent in all packets, and packets are accepted if they are properly signed by either of the keys. At that point, the old key is removed.

In order to support incremental deployment and key rotation, implementations SHOULD support an interface configuration in which they send authenticated packets but accept all packets, and SHOULD
allow changing the set of keys associated with an interface without a restart.

Appendix B. Changes from previous versions

[RFC Editor: please remove this section before publication.]

B.1. Changes since draft-ietf-babel-hmac-00

- Changed the title.
- Removed the appendix about the packet trailer, this is now in rfc6126bis.
- Removed the appendix with implicit indices.
- Clarified the definitions of acronyms.
- Limited the size of nonces and indices.

B.2. Changes since draft-ietf-babel-hmac-01

- Made BLAKE2s a recommended HMAC algorithm.
- Added requirement to expire per-neighbour crypto state.

B.3. Changes since draft-ietf-babel-hmac-02

- Clarified that PCs are 32-bit unsigned integers.
- Clarified that indices and nonces are of arbitrary size.
- Added reference to RFC 4086.

B.4. Changes since draft-ietf-babel-hmac-03

- Use the TLV values allocated by IANA.
- Fixed an issue with packets that contain a successful challenge reply: they should be accepted before checking the PC value.
- Clarified that keys are the exact value of the HMAC hash size, and not subject to preprocessing; this makes management more deterministic.
Authors’ Addresses

Clara Do
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France
Email: clarado_perso@yahoo.fr

Weronika Kolodziejak
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France
Email: weronika.kolodziejak@gmail.com

Juliusz Chroboczek
IRIF, University of Paris-Diderot
Case 7014
75205 Paris Cedex 13
France
Email: jch@irif.fr
HMAC authentication for the Babel routing protocol
draft-ietf-babel-hmac-07

Abstract

This document describes a cryptographic authentication mechanism for
the Babel routing protocol that has provisions for replay avoidance.
This document obsoletes RFC 7298.

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 https://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 December 22, 2019.

Copyright Notice

Copyright (c) 2019 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
(https://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
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
By default, the Babel routing protocol trusts the information contained in every UDP datagram that it receives on the Babel port. An attacker can redirect traffic to itself or to a different node in the network, causing a variety of potential issues. In particular, an attacker might:

- spoof a Babel packet, and redirect traffic by announcing a smaller metric, a larger seqno, or a longer prefix;
o spoof a malformed packet, which could cause an insufficiently robust implementation to crash or interfere with the rest of the network;

o replay a previously captured Babel packet, which could cause traffic to be redirected or otherwise interfere with the network.

Protecting a Babel network is challenging due to the fact that the Babel protocol uses both unicast and multicast communication. One possible approach, used notably by the Babel over Datagram Transport Layer Security (DTLS) protocol [I-D.ietf-babel-dtls], is to use unicast communication for all semantically significant communication, and then use a standard unicast security protocol to protect the Babel traffic. In this document, we take the opposite approach: we define a cryptographic extension to the Babel protocol that is able to protect both unicast and multicast traffic, and thus requires very few changes to the core protocol.

1.1. Applicability

The protocol defined in this document assumes that all interfaces on a given link are equally trusted and share a small set of symmetric keys (usually just one, and two during key rotation). The protocol is inapplicable in situations where asymmetric keying is required, where the trust relationship is partial, or where large numbers of trusted keys are provisioned on a single link at the same time.

This protocol supports incremental deployment (where an insecure Babel network is made secure with no service interruption), and it supports graceful key rotation (where the set of keys is changed with no service interruption).

This protocol does not require synchronised clocks, it does not require persistently monotonic clocks, and it does not require persistent storage except for what might be required for storing cryptographic keys.

1.2. Assumptions and security properties

The correctness of the protocol relies on the following assumptions:

o that the Hashed Message Authentication Code (HMAC) being used is invulnerable to pre-image attacks, i.e., that an attacker is unable to generate a packet with a correct HMAC;

o that a node never generates the same index or nonce twice over the lifetime of a key.
The first assumption is a property of the HMAC being used. The second assumption can be met either by using a robust random number generator [RFC4086] and sufficiently large indices and nonces, by using a reliable hardware clock, or by rekeying whenever a collision becomes likely.

If the assumptions above are met, the protocol described in this document has the following properties:

- **o** it is invulnerable to spoofing: any packet accepted as authentic is the exact copy of a packet originally sent by an authorised node;

- **o** locally to a single node, it is invulnerable to replay: if a node has previously accepted a given packet, then it will never again accept a copy of this packet or an earlier packet from the same sender;

- **o** among different nodes, it is only vulnerable to immediate replay: if a node A has accepted a packet from C as valid, then a node B will only accept a copy of that packet as authentic if B has accepted an older packet from C and B has received no later packet from C.

While this protocol makes serious efforts to mitigate the effects of a denial of service attack, it does not fully protect against such attacks.

1.3. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Conceptual overview of the protocol

When a node B sends out a Babel packet through an interface that is configured for HMAC cryptographic protection, it computes one or more HMACs which it appends to the packet. When a node A receives a packet over an interface that requires HMAC cryptographic protection, it independently computes a set of HMACs and compares them to the HMACs appended to the packet; if there is no match, the packet is discarded.

In order to protect against replay, B maintains a per-interface 32-bit integer known as the "packet counter" (PC). Whenever B sends
a packet through the interface, it embeds the current value of the PC within the region of the packet that is protected by the HMACs and increases the PC by at least one. When A receives the packet, it compares the value of the PC with the one contained in the previous packet received from B, and unless it is strictly greater, the packet is discarded.

By itself, the PC mechanism is not sufficient to protect against replay. Consider a peer A that has no information about a peer B (e.g., because it has recently rebooted). Suppose that A receives a packet ostensibly from B carrying a given PC; since A has no information about B, it has no way to determine whether the packet is freshly generated or a replay of a previously sent packet.

In this situation, A discards the packet and challenges B to prove that it knows the HMAC key. It sends a "challenge request", a TLV containing a unique nonce, a value that has never been used before and will never be used again. B replies to the challenge request with a "challenge reply", a TLV containing a copy of the nonce chosen by A, in a packet protected by HMAC and containing the new value of B’s PC. Since the nonce has never been used before, B’s reply proves B’s knowledge of the HMAC key and the freshness of the PC.

By itself, this mechanism is safe against replay if B never resets its PC. In practice, however, this is difficult to ensure, as persistent storage is prone to failure, and hardware clocks, even when available, are occasionally reset. Suppose that B resets its PC to an earlier value, and sends a packet with a previously used PC n. A challenges B, B successfully responds to the challenge, and A accepts the PC equal to n + 1. At this point, an attacker C may send a replayed packet with PC equal to n + 2, which will be accepted by A.

Another mechanism is needed to protect against this attack. In this protocol, every PC is tagged with an "index", an arbitrary string of octets. Whenever B resets its PC, or whenever B doesn’t know whether its PC has been reset, it picks an index that it has never used before (either by drawing it randomly or by using a reliable hardware clock) and starts sending PCs with that index. Whenever A detects that B has changed its index, it challenges B again.

With this additional mechanism, this protocol is invulnerable to replay attacks (see Section 1.2 above).
3. Data Structures

Every Babel node maintains a set of conceptual data structures described in Section 3.2 of [RFC6126bis]. This protocol extends these data structures as follows.

3.1. The Interface Table

Every Babel node maintains an interface table, as described in Section 3.2.3 [RFC6126bis]. Implementations of this protocol MUST allow each interface to be provisioned with a set of one or more HMAC keys and the associated HMAC algorithms (see Section 4.1). In order to allow incremental deployment of this protocol (see Appendix A), implementations SHOULD allow an interface to be configured in a mode in which it participates in the HMAC authentication protocol but accepts packets that are not authenticated.

This protocol extends each entry in this table that is associated with an interface on which HMAC authentication has been configured with two new pieces of data:

- a set of one or more HMAC keys, each associated with a given HMAC algorithm; the length of each key is exactly the hash size of the associated HMAC algorithm (i.e., the key is not subject to the preprocessing described in Section 2 of [RFC2104]);

- a pair (Index, PC), where Index is an arbitrary string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer.

We say that an index is fresh when it has never been used before with any of the keys currently configured on the interface. The Index field is initialised to a fresh index, for example by drawing a random string of sufficient length, and the PC is initialised to an arbitrary value (typically 0).

3.2. The Neighbour Table

Every Babel node maintains a neighbour table, as described in Section 3.2.4 of [RFC6126bis]. This protocol extends each entry in this table with two new pieces of data:

- a pair (Index, PC), where Index is a string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer;

- a Nonce, which is an arbitrary string of 0 to 192 octets, and an associated challenge expiry timer.
The Index and PC are initially undefined, and are managed as described in Section 4.3. The Nonce and expiry timer are initially undefined, and used as described in Section 4.3.1.1.

4. Protocol Operation

4.1. HMAC computation

A Babel node computes the HMAC of a Babel packet as follows.

First, the node builds a pseudo-header that will participate in HMAC computation but will not be sent. If the packet was carried over IPv6, the pseudo-header has the following format:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If the packet was carried over IPv4, the pseudo-header has the following format:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Src address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Src port            |                               | Dest address     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |           Dest port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If the packet was carried over IPv4, the pseudo-header has the following format:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Src address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Src port                          | Dest address     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Dest port                          |                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Dest port                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Fields:

Src address  The source IP address of the packet.

Src port     The source UDP port number of the packet.

Dest address The destination IP address of the packet.

Src port     The destination UDP port number of the packet.

The node takes the concatenation of the pseudo-header and the packet including the packet header but excluding the packet trailer (from octet 0 inclusive up to (Body Length + 4) exclusive) and computes an HMAC with one of the implemented hash algorithms. Every implementation MUST implement HMAC-SHA256 as defined in [RFC6234] and Section 2 of [RFC2104], SHOULD implement keyed BLAKE2s [RFC7693], and MAY implement other HMAC algorithms.

4.2. Packet Transmission

A Babel node might delay actually sending TLVs by a small amount, in order to aggregate multiple TLVs in a single packet up to the interface MTU (Section 4 of [RFC6126bis]). For an interface on which HMAC protection is configured, the TLV aggregation logic MUST take into account the overhead due to PC TLVs (one in each packet) and HMAC TLVs (one per configured key).

Before sending a packet, the following actions are performed:

- a PC TLV containing the PC and Index associated with the outgoing interface MUST be appended to the packet body; the PC MUST be incremented by a strictly positive amount (typically just 1); if the PC overflows, a fresh index MUST be generated (as defined in Section 3.1); a node MUST NOT include multiple PC TLVs in a single packet;

- for each key configured on the interface, an HMAC is computed as specified in Section 4.1 above, and stored in an HMAC TLV that MUST be appended to the packet trailer (see Section 4.2 of [RFC6126bis]).

4.3. Packet Reception

When a packet is received on an interface that is configured for HMAC protection, the following steps are performed before the packet is passed to normal processing:
First, the receiver checks whether the trailer of the received packet carries at least one HMAC TLV; if not, the packet MUST be immediately dropped and processing stops. Then, for each key configured on the receiving interface, the receiver computes the HMAC of the packet. It then compares every generated HMAC against every HMAC included in the packet; if there is at least one match, the packet passes the HMAC test; if there is none, the packet MUST be silently dropped and processing stops at this point. In order to avoid memory exhaustion attacks, an entry in the Neighbour Table MUST NOT be created before the HMAC test has passed successfully. The HMAC of the packet MUST NOT be computed for each HMAC TLV contained in the packet, but only once for each configured key.

The packet body is then parsed a first time. During this "preparse" phase, the packet body is traversed and all TLVs are ignored except PC TLVs, Challenge Requests and Challenge Replies. When a PC TLV is encountered, the enclosed PC and Index are saved for later processing; if multiple PCs are found (which should not happen, see Section 4.2 above), only the first one is processed, the remaining ones MUST be silently ignored. If a Challenge Request is encountered, a Challenge Reply MUST be scheduled, as described in Section 4.3.1.2. If a Challenge Reply is encountered, it is tested for validity as described in Section 4.3.1.3 and a note is made of the result of the test.

The preparse phase above has yielded two pieces of data: the PC and Index from the first PC TLV, and a bit indicating whether the packet contains a successful Challenge Reply. If the packet does not contain a PC TLV, the packet MUST be dropped and processing stops at this point. If the packet contains a successful Challenge Reply, then the PC and Index contained in the PC TLV MUST be stored in the Neighbour Table entry corresponding to the sender (which may need to be created at this stage), and the packet is accepted.

Otherwise, if there is no entry in the Neighbour Table corresponding to the sender, or if such an entry exists but contains no Index, or if the Index it contains is different from the Index contained in the PC TLV, then a challenge MUST be sent as described in Section 4.3.1.1, the packet MUST be dropped, and processing stops at this stage.

At this stage, the packet contains no successful challenge reply and the Index contained in the PC TLV is equal to the Index in the Neighbour Table entry corresponding to the sender. The receiver compares the received PC with the PC contained in the Neighbour Table; if the received PC is smaller or equal than the PC
contained in the Neighbour Table, the packet MUST be dropped and processing stops (no challenge is sent in this case, since the mismatch might be caused by harmless packet reordering on the link). Otherwise, the PC contained in the Neighbour Table entry is set to the received PC, and the packet is accepted.

In the algorithm described above, challenge requests are processed and challenges are sent before the PC/Index pair is verified against the neighbour table. This simplifies the implementation somewhat (the node may simply schedule outgoing requests as it walks the packet during the preparse phase), but relies on the rate-limiting described in Section 4.3.1.1 to avoid sending too many challenges in response to replayed packets. As an optimisation, a node MAY ignore all challenge requests contained in a packet except the last one, and it MAY ignore a challenge request in the case where it it contained in a packet with an Index that matches the one in the Neighbour Table and a PC that is smaller or equal to the one contained in the Neighbour Table. Since it is still possible to replay a packet with an obsolete Index, the rate-limiting described in Section 4.3.1.1 is required even if this optimisation is implemented.

The same is true of challenge replies. However, since validating a challenge reply is extremely cheap (it’s just a bitwise comparison of two strings of octets), a similar optimisation for challenge replies is not worthwhile.

After the packet has been accepted, it is processed as normal, except that any PC, Challenge Request and Challenge Reply TLVs that it contains are silently ignored.

4.3.1. Challenge Requests and Replies

During the preparse stage, the receiver might encounter a mismatched Index, to which it will react by scheduling a Challenge Request. It might encounter a Challenge Request TLV, to which it will reply with a Challenge Reply TLV. Finally, it might encounter a Challenge Reply TLV, which it will attempt to match with a previously sent Challenge Request TLV in order to update the Neighbour Table entry corresponding to the sender of the packet.

4.3.1.1. Sending challenges

When it encounters a mismatched Index during the preparse phase, a node picks a nonce that it has never used with any of the keys currently configured on the relevant interface, for example by drawing a sufficiently large random string of bytes or by consulting a strictly monotonic hardware clock. It MUST then store the nonce in the entry of the Neighbour Table associated to the neighbour (the
A node MAY aggregate a Challenge Request with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the neighbour, it MAY send the buffered TLVs in the same packet as the Challenge Request. However, it MUST arrange for the Challenge Request to be sent in a timely manner, as any packets received from that neighbour will be silently ignored until the challenge completes.

Since a challenge may be prompted by a packet replayed by an attacker, a node MUST impose a rate limitation to the challenges it sends; the limit SHOULD default to one challenge request every 300ms, and MAY be configurable.

4.3.1.2. Replying to challenges

When it encounters a Challenge Request during the preparse phase, a node constructs a Challenge Reply TLV by copying the Nonce from the Challenge Request into the Challenge Reply. It MUST then send the Challenge Reply to the unicast address from which the Challenge Request was sent.

A node MAY aggregate a Challenge Reply with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the sender of the Challenge Request, it MAY send the buffered TLVs in the same packet as the Challenge Reply. However, it MUST arrange for the Challenge Reply to be sent in a timely manner (within a few seconds), and SHOULD NOT send any other packets over the same interface before sending the Challenge Reply, as those would be dropped by the challenger.

A challenge sent to a multicast address MUST be silently ignored.

4.3.1.3. Receiving challenge replies

When it encounters a Challenge Reply during the preparse phase, a node consults the Neighbour Table entry corresponding to the neighbour that sent the Challenge Reply. If no challenge is in progress, i.e., if there is no Nonce stored in the Neighbour Table entry or the Challenge timer has expired, the Challenge Reply MUST be silently ignored and the challenge has failed.

Otherwise, the node compares the Nonce contained in the Challenge Reply with the Nonce contained in the Neighbour Table entry. If the
two are equal (they have the same length and content), then the challenge has succeeded; otherwise, the challenge has failed.

4.4. Expiring per-neighbour state

The per-neighbour (Index, PC) pair is maintained in the neighbour table, and is normally discarded when the neighbour table entry expires. Implementations MUST ensure that an (Index, PC) pair is discarded within a finite time since the last time a packet has been accepted. In particular, unsuccessful challenges MUST NOT prevent an (Index, PC) pair from being discarded for unbounded periods of time.

A possible implementation strategy for implementations that use a Hello history (Appendix A of [RFC6126bis]) is to discard the (Index, PC) pair whenever the Hello history becomes empty. Another implementation strategy is to use a timer that is reset whenever a packet is accepted, and to discard the (Index, PC) pair whenever the timer expires. If the latter strategy is being used, the timer SHOULD default to a value of 5 min, and MAY be configurable.

5. Packet Format

5.1. HMAC TLV

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 16   |    Length     |     HMAC...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type Set to 16 to indicate an HMAC TLV.

Length The length of the body, in octets, exclusive of the Type and Length fields. The length of the body depends on the HMAC algorithm being used.

HMAC The body contains the HMAC of the packet, computed as described in Section 4.1.

This TLV is allowed in the packet trailer (see Section 4.2 of [RFC6126bis]), and MUST be ignored if it is found in the packet body.
5.2. PC TLV

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 17   |    Length     |             PC                |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |            Index...
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type  Set to 17 to indicate a PC TLV.
Length  The length of the body, in octets, exclusive of the Type and Length fields.
PC  The Packet Counter (PC), a 32-bit (4 octet) unsigned integer which is increased with every packet sent over this interface. A fresh index (as defined in Section 3.1) MUST be generated whenever the PC overflows.
Index  The sender’s Index, an opaque string of 0 to 32 octets.

Indices are limited to a size of 32 octets: a node MUST NOT send a TLV with an index of size strictly larger than 32 octets, and a node MAY ignore a PC TLV with an index of length strictly larger than 32 octets. Indices of length 0 are valid: if a node has reliable stable storage and the packet counter never overflows, then only one index is necessary, and the value of length 0 is the canonical choice.

5.3. Challenge Request TLV

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 18   |    Length     |     Nonce... |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type  Set to 18 to indicate a Challenge Request TLV.
Length  The length of the body, in octets, exclusive of the Type and Length fields.
Nonce  The nonce uniquely identifying the challenge, an opaque string of 0 to 192 octets.
Nonces are limited to a size of 192 octets: a node MUST NOT send a Challenge Request TLV with a nonce of size strictly larger than 192 octets, and a node MAY ignore a nonce that is of size strictly larger than 192 octets. Nonces of length 0 are valid: if a node has reliable stable storage, then it may use a sequential counter for generating nonces which get encoded in the minimum number of octets required; the value 0 is then encoded as the string of length 0.

5.4. Challenge Reply TLV

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 19   |    Length     |     Nonce...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

- **Type**: Set to 19 to indicate a Challenge Reply TLV.
- **Length**: The length of the body, in octets, exclusive of the Type and Length fields.
- **Nonce**: A copy of the nonce contained in the corresponding challenge request.

6. Security Considerations

This document defines a mechanism that provides basic security properties for the Babel routing protocol. The scope of this protocol is strictly limited: it only provides authentication (we assume that routing information is not confidential), it only supports symmetric keying, and it only allows for the use of a small number of symmetric keys on every link. Deployments that need more features, e.g., confidentiality or asymmetric keying, should use a more featureful security mechanism such as the one described in [I-D.ietf-babel-dtls].

This mechanism relies on two assumptions, as described in Section 1.2. First, it assumes that the hash being used is invulnerable to pre-image attacks (Section 1.1 of [RFC6039]); at the time of writing, SHA-256, which is mandatory to implement (Section 4.1), is believed to be safe against practical attacks.

Second, it assumes that indices and nonces are generated uniquely over the lifetime of a key used for HMAC computation (more precisely, indices must be unique for a given (key, source) pair, and nonces must be unique for a given (key, source, destination) triple). This
property can be satisfied either by using a cryptographically secure random number generator to generate indices and nonces that contain enough entropy (64-bit values are believed to be large enough for all practical applications), or by using a reliably monotonic hardware clock. If uniqueness cannot be guaranteed (e.g., because a hardware clock has been reset), then rekeying is necessary.

The expiry mechanism mandated in Section 4.4 is required to prevent an attacker from delaying an authentic packet by an unbounded amount of time. If an attacker is able to delay the delivery of a packet (e.g., because it is located at a layer 2 switch), then the packet will be accepted as long as the corresponding (Index, PC) pair is present at the receiver. If the attacker is able to cause the (Index, PC) pair to persist for arbitrary amounts of time (e.g., by repeatedly causing failed challenges), then it is able to delay the packet by arbitrary amounts of time, even after the sender has left the network.

While it is probably not possible to be immune against denial of service (DoS) attacks in general, this protocol includes a number of mechanisms designed to mitigate such attacks. In particular, reception of a packet with no correct HMAC creates no local state whatsoever (Section 4.3). Reception of a replayed packet with correct hash, on the other hand, causes a challenge to be sent; this is mitigated somewhat by requiring that challenges be rate-limited.

At first sight, sending a challenge requires retaining enough information to validate the challenge reply. However, the nonce included in a challenge request and echoed in the challenge reply can be fairly large (up to 192 octets), which should in principle permit encoding the per-challenge state as a secure "cookie" within the nonce itself.

7. IANA Considerations

IANA has allocated the following values in the Babel TLV Types registry:
8. Acknowledgments

The protocol described in this document is based on the original HMAC protocol defined by Denis Ovsienko [RFC7298]. The use of a pseudo-header was suggested by David Schinazi. The use of an index to avoid replay was suggested by Markus Stenberg. The authors are also indebted to Donald Eastlake, Toke Holiland-Jorgensen, Florian Horn, Dave Taht and Martin Vigoureux.

9. References

9.1. Normative References


Appendix A. Incremental deployment and key rotation

This protocol supports incremental deployment (transitioning from an insecure network to a secured network with no service interruption) and key rotation (transitioning from a set of keys to a different set of keys).

In order to perform incremental deployment, the nodes in the network are first configured in a mode where packets are sent with authentication but not checked on reception. Once all the nodes in the network are configured to send authenticated packets, nodes are reconfigured to reject unauthenticated packets.

In order to perform key rotation, the new key is added to all the nodes; once this is done, both the old and the new key are sent in all packets, and packets are accepted if they are properly signed by either of the keys. At that point, the old key is removed.

In order to support incremental deployment and key rotation, implementations SHOULD support an interface configuration in which they send authenticated packets but accept all packets, and SHOULD
allow changing the set of keys associated with an interface without a restart.

Appendix B. Changes from previous versions

[RFC Editor: please remove this section before publication.]

B.1. Changes since draft-ietf-babel-hmac-00
   o Changed the title.
   o Removed the appendix about the packet trailer, this is now in rfc6126bis.
   o Removed the appendix with implicit indices.
   o Clarified the definitions of acronyms.
   o Limited the size of nonces and indices.

B.2. Changes since draft-ietf-babel-hmac-01
   o Made BLAKE2s a recommended HMAC algorithm.
   o Added requirement to expire per-neighbour crypto state.

B.3. Changes since draft-ietf-babel-hmac-02
   o Clarified that PCs are 32-bit unsigned integers.
   o Clarified that indices and nonces are of arbitrary size.
   o Added reference to RFC 4086.

B.4. Changes since draft-ietf-babel-hmac-03
   o Use the TLV values allocated by IANA.
   o Fixed an issue with packets that contain a successful challenge reply: they should be accepted before checking the PC value.
   o Clarified that keys are the exact value of the HMAC hash size, and not subject to preprocessing; this makes management more deterministic.
B.5. Changes since draft-ietf-babel-hmac-04
   o Use normative language in more places.

B.6. Changes since draft-ietf-babel-hmac-05
   o Do not update RFC 6126bis.
   o Clarify that indices and nonces of length 0 are valid.
   o Clarify that multiple PC TLVs in a single packet are not allowed.
   o Allow discarding challenge requests when they carry an old PC.

B.7. Changes since draft-ietf-babel-hmac-06
   o Do not update RFC 6126bis, for real this time.

Authors’ Addresses

Clara Do
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France

Email: clarado_perso@yahoo.fr

Weronika Kolodziejak
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France

Email: weronika.kolodziejak@gmail.com

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

Email: jch@irif.fr
Babel Information Model
draft-ietf-babel-information-model-04

Abstract

This Babel Information Model can be used to create data models under various data modeling regimes (e.g., YANG). It allows a Babel implementation (via a management protocol such as NETCONF) to report on its current state and may allow some limited configuration of protocol constants.

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 https://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 April 25, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1.  Introduction

Babel is a loop-avoiding distance-vector routing protocol defined in
[I-D.ietf-babel-rfc6126bis].  [I-D.ietf-babel-hmac] defines a
security mechanism that allows Babel messages to be cryptographically
authenticated, and [I-D.ietf-babel-dtls] defines a security mechanism
that allows Babel messages to be encrypted.  This document describes
an information model for Babel (including implementations using one
of these security mechanisms) that can be used to create management
protocol data models (such as a NETCONF [RFC6241] YANG [RFC7950] data
model).

Due to the simplicity of the Babel protocol, most of the information
model is focused on reporting Babel protocol operational state, and
very little of that is considered mandatory to implement (contingent
on a management protocol with Babel support being implemented).  Some
parameters may be configurable.  However, it is up to the Babel
implementation whether to allow any of these to be configured within
its implementation.  Where the implementation does not allow
configuration of these parameters, it may still choose to expose them
as read-only.
The Information Model is presented using a hierarchical structure. This does not preclude a data model based on this Information Model from using a referential or other structure.

1.1. Requirements Language

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] and updated by [RFC8174].

1.2. Notation

This document uses a programming language-like notation to define the properties of the objects of the information model. An optional property is enclosed by square brackets, [ ], and a list property is indicated by two numbers in angle brackets, <m..n>, where m indicates the minimal number of list elements, and n indicates the maximum number of list elements. The symbol * for n means there are no defined limits on the number of list elements. Each parameter and object includes an indication of "ro" or "rw". "ro" means the parameter or object is read-only. "rw" means it is read-write. For an object, read-write means instances of the object can be created or deleted. If an implementation is allowed to choose to implement a "rw" parameter as read-only, this is noted in the parameter description.

The object definitions use base types that are defined as follows:

- **binary**: A binary string (sequence of octets).
- **boolean**: A type representing a boolean value.
- **counter**: A non-negative integer that monotonically increases. Counters may have discontinuities and they are not expected to persist across restarts.
- **credentials**: An opaque type representing credentials needed by a cryptographic mechanism to secure communication. Data models must expand this opaque type as needed and required by the security protocols utilized.
- **datetime**: A type representing a date and time using the Gregorian calendar. The datetime format MUST conform to RFC 3339 [RFC3339].
- **int**: A type representing signed or unsigned integer numbers. This information model does not define a precision nor
does it make a distinction between signed and unsigned number ranges.

ip-address  A type representing an IP address. This type supports both IPv4 and IPv6 addresses.

string     A type representing a human-readable string consisting of a (possibly restricted) subset of Unicode and ISO/IEC 10646 [ISO.10646] characters.

uri        A type representing a Uniform Resource Identifier as defined in STD 66 [RFC3986].

2. Overview

The Information Model is hierarchically structured as follows:

- information object
  - includes implementation version, router id, this node seqno, enable flag parameters, supported security mechanisms
  - constants object (exactly one per information object)
    - includes UDP port and optional multicast group parameters
- interfaces object
  - includes interface reference, Hello seqno and intervals, update interval, link type, metric computation parameters
  - neighbors object
    - includes neighbor IP address, Hello history, cost parameters
- security object (per interface)
  - includes enable flag, self credentials (credential object), trusted credentials (credential object)
- security object (common to all interfaces)
  - includes enable flag, self credentials (credential object), trusted credentials (credential object)
- routes object
  - includes route prefix, source router, reference to advertising neighbor, metric, sequence number, whether route is feasible, whether route is selected

Most parameters are read-only. Following is a list of the parameters that are not required to be read-only:

- enable/disable Babel
- Constant: UDP port
- Constant: IPv6 multicast group
3. The Information Model

3.1. Definition of babel-information-obj

```plaintext
object {
    string               ro babel-implementation-version;
    boolean              rw babel-enable;
    binary               ro babel-self-router-id;
    string               ro babel-supported-link-types<1..*>;
    [int                  ro babel-self-seqno;]
    string               ro babel-metric-comp-algorithms<1..*>;
    string               ro babel-security-supported<0..*>;
    babel-constants-obj  ro babel-constants;
    babel-interfaces-obj ro babel-interfaces<0..*>;
    babel-routes-obj    ro babel-routes<0..*>;
    babel-security-obj   ro babel-security<0..*>;
} babel-information-obj;
```

**babel-implementation-version**: The name and version of this implementation of the Babel protocol.

**babel-enable**: When written, it configures whether the protocol should be enabled (true) or disabled (false). A read from the running or intended datastore indicates the configured administrative value of whether the protocol is enabled (true) or not (false). A read from the operational datastore indicates whether the protocol is enabled.
actually running (true) or not (i.e., it indicates the operational state of the protocol). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-self-router-id: The router-id used by this instance of the Babel protocol to identify itself. [I-D.ietf-babel-rfc6126bis] describes this as an arbitrary string of 8 octets.

babel-supported-link-types: Lists the set of link types supported by this instance of Babel. Valid enumeration values are defined in the Babel Link Types registry (see Section 7).

babel-self-seqno: The current sequence number included in route updates for routes originated by this node.

babel-metric-comp-algorithms: List of supported cost computation algorithms. Possible values include "k-out-of-j", and "ETX".

babel-security-supported: List of supported security mechanisms. As Babel security mechanisms are defined, they will need to indicate what enumeration value is to be used to represent them in this parameter.

babel-constants: A babel-constants-obj object.

babel-interfaces: A set of babel-interface-obj objects.

babel-security: A babel-security-obj object that applies to all interfaces. If this object is implemented, it allows a security mechanism to be enabled or disabled in a manner that applies to all Babel messages on all interfaces.

babel-routes: A set of babel-route-obj objects. Contains the routes known to this node.

3.2. Definition of babel-constants-obj

object {
    int rw babel-udp-port;
    [ip-address rw babel-mcast-group;]
} babel-constants-obj;

babel-udp-port: UDP port for sending and listening for Babel messages. Default is 6696. An implementation MAY choose to expose this parameter as read-only ("ro").
babel-mcast-group: Multicast group for sending and listening to multicast announcements on IPv6. Default is ff02:0:0:0:0:0:1:6. An implementation MAY choose to expose this parameter as read-only ("ro").

3.3. Definition of babel-interfaces-obj

object {
    string ro babel-interface-reference;
    [boolean rw babel-interface-enable;]
    int rw babel-link-type;
    string ro babel-interface-metric-algorithm;
    [int ro babel-mcast-hello-seqno;]
    [int ro babel-mcast-hello-interval;]
    [int ro babel-update-interval;]
    [boolean rw babel-message-log-enable;]
    [babel-log-obj ro babel-message-log<0..*>;]
    [babel-neighbors-obj ro babel-neighbors<0..*>;]
    [babel-security-obj ro babel-interface-security<0..*>;]
} babel-interfaces-obj;

babel-interface-reference: Reference to an interface object as defined by the data model (e.g., YANG [RFC7950], BBF [TR-181]). Data model is assumed to allow for referencing of interface objects which may be at any layer (physical, Ethernet MAC, IP, tunnelled IP, etc.). referencing syntax will be specific to the data model. If there is no set of interface objects available, this should be a string that indicates the interface name used by the underlying operating system.

babel-interface-enable: When written, it configures whether the protocol should be enabled (true) or disabled (false) on this interface. A read from the running or intended datastore indicates the configured administrative value of whether the protocol is enabled (true) or not (false). A read from the operational datastore indicates whether the protocol is actually running (true) or not (i.e., it indicates the operational state of the protocol). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-link-type: Indicates the type of link. Valid enumeration values are identified in Babel Link Types registry. An implementation MAY choose to expose this parameter as read-only ("ro").
babel-interface-metric-algorithm: Indicates the metric computation algorithm used on this interface. The value MUST be one of those listed in the babel-information-obj babel-metric-comp-algorithms parameter.

babel-mcast-hello-seqno: The current sequence number in use for multicast hellos sent on this interface.

babel-mcast-hello-interval: The current interval in use for multicast hellos sent on this interface.

babel-update-interval: The current interval in use for all updates (multicast and unicast) sent on this interface.

babel-message-log-enable: When written, it configures whether logging should be enabled (true) or disabled (false). A read from the running or intended datastore indicates the configured administrative value of whether logging is enabled (true) or not (false). A read from the operational datastore indicates whether logging is actually running (true) or not (i.e., it indicates the operational state). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-message-log: Log entries that have timestamp of a received Babel message and the entire received Babel message (including Ethernet frame and IP headers, if possible). An implementation must restrict the size of this log, but how and what size is implementation-specific. If this log is implemented, a mechanism to clear it SHOULD be provided.

babel-neighbors: A set of babel-neighbors-obj objects.

babel-interface-security: A babel-security-obj object that applies to this interface. If implemented, this allows security to be enabled only on specific interfaces or allows different security mechanisms to be enabled on different interfaces.

3.4. Definition of babel-neighbors-obj
object {
    ip-address        ro babel-neighbor-address;
 [binary            ro babel-hello-mcast-history;]
 [binary            ro babel-hello-ucast-history;]
 int               ro babel-txcost;
 int               ro babel-exp-mcast-hello-seqno;
 int               ro babel-exp-ucast-hello-seqno;
 [int               ro babel-ucast-hello-seqno;]
 [int               ro babel-ucast-hello-interval;]
 [int               ro babel-rxcost]
 [int               ro babel-cost]
} babel-neighbors-obj;

babel-neighbor-address: IPv4 or IPv6 address the neighbor sends messages from

babel-hello-mcast-history: The multicast Hello history of whether or not the multicast Hello messages prior to babel-exp-mcast-hello-seqno were received. A binary sequence where the most recently received Hello is expressed as a "1" placed in the left-most bit, with prior bits shifted right (and "0" bits placed between prior Hello bits and most recent Hello for any not-received Hellos). This value should be displayed using hex digits ([0-9a-fA-F]). See [I-D.ietf-babel-rfc6126bis], section A.1.

babel-hello-ucast-history: The unicast Hello history of whether or not the unicast Hello messages prior to babel-exp-ucast-hello-seqno were received. A binary sequence where the most recently received Hello is expressed as a "1" placed in the left-most bit, with prior bits shifted right (and "0" bits placed between prior Hello bits and most recent Hello for any not-received Hellos). This value should be displayed using hex digits ([0-9a-fA-F]). See [I-D.ietf-babel-rfc6126bis], section A.1.

babel-txcost: Transmission cost value from the last IHU packet received from this neighbor, or maximum value (infinity) to indicate the IHU hold timer for this neighbor has expired. See [I-D.ietf-babel-rfc6126bis], section 3.4.2.

babel-exp-mcast-hello-seqno: Expected multicast Hello sequence number of next Hello to be received from this neighbor. If multicast Hello messages are not expected, or processing of multicast messages is not enabled, this MUST be 0.

babel-exp-ucast-hello-seqno: Expected unicast Hello sequence number of next Hello to be received from this neighbor. If unicast Hello messages are not expected, or processing of unicast messages is not enabled, this MUST be 0.
babel-ucast-hello-seqno: The current sequence number in use for unicast hellos sent to this neighbor.

babel-ucast-hello-interval: The current interval in use for unicast hellos sent to this neighbor.

babel-rxcost: Reception cost calculated for this neighbor. This value is usually derived from the Hello history, which may be combined with other data, such as statistics maintained by the link layer. The rxcost is sent to a neighbor in each IHU. See [I-D.ietf-babel-rfc6126bis], section 3.4.3.

babel-cost: Link cost is computed from the values maintained in the neighbor table: the statistics kept in the neighbor table about the reception of Hellos, and the txcost computed from received IHU packets.

3.5. Definition of babel-security-obj

object {
    string     ro babel-security-mechanism
    boolean    rw babel-security-enable;
    babel-credential-obj ro babel-security-self-cred<0..*>;
    babel-credential-obj ro babel-security-trust<0..*>;
    [boolean    rw babel-credvalid-log-enable;]
    [babel-log-obj ro babel-credvalid-log<0..*>;]
} babel-security-obj;

babel-security-mechanism: The name of the security mechanism this object instance is about. The value MUST be the same as one of the enumerations listed in the babel-security-supported parameter.

babel-security-enable: When written, it configures whether this security mechanism should be enabled (true) or disabled (false). A read from the running or intended datastore indicates the configured administrative value of whether this security mechanism is enabled (true) or not (false). A read from the operational datastore indicates whether this security mechanism is actually running (true) or not (i.e., it indicates the operational state). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-security-self-cred: Credentials this router presents to participate in the enabled security mechanism. Any private key component of a credential MUST NOT be readable. Adding and deleting credentials MAY be allowed.
babel-security-trust: A set of babel-credential-obj objects that identify the credentials of routers whose Babel messages may be trusted or of a certificate authority (CA) whose signing of a router’s credentials implies the router credentials can be trusted, in the context of this security mechanism. How a security mechanism interacts with this list is determined by the mechanism. A security algorithm may do additional validation of credentials, such as checking validity dates or revocation lists, so presence in this list may not be sufficient to determine trust. Adding and deleting credentials MAY be allowed.

babel-credvalid-log-enable: When written, it configures whether logging should be enabled (true) or disabled (false). A read from the running or intended datastore indicates the configured administrative value of whether logging is enabled (true) or not (false). A read from the operational datastore indicates whether logging is actually running (true) or not (i.e., it indicates the operational state). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-credvalid-log: Log entries that have the timestamp a message containing credentials used for peer authentication (e.g., DTLS Server Hello) was received on a Babel port, and the entire received message (including Ethernet frame and IP headers, if possible). An implementation must restrict the size of this log, but how and what size is implementation-specific. If this log is implemented, a mechanism to clear it SHOULD be provided.

3.6. Definition of babel-routes-obj

object {
  ip-address ro babel-route-prefix;
  int ro babel-route-prefix-length;
  binary ro babel-route-router-id;
  string ro babel-route-neighbor;
  [int ro babel-route-received-metric;]
  [int ro babel-route-calculated-metric;]
  int ro babel-route-seqno;
  ip-address ro babel-route-next-hop;
  boolean ro babel-route-feasible;
  boolean ro babel-route-selected;
} babel-routes-obj;

babel-route-prefix: Prefix (expressed in IP address format) for which this route is advertised.
babel-route-prefix-length: Length of the prefix for which this route is advertised.

babel-route-router-id: router-id of the source router for which this route is advertised.

babel-route-neighbor: Reference to the babel-neighbors entry for the neighbor that advertised this route.

babel-route-received-metric: The metric with which this route was advertised by the neighbor, or maximum value (infinity) to indicate the route was recently retracted and is temporarily unreachable (see Section 3.5.5 of [I-D.ietf-babel-rfc6126bis]). This metric will be 0 (zero) if the route was not received from a neighbor but was generated through other means. Either babel-route-calculated-metric or babel-route-received-metric MUST be provided.

babel-route-calculated-metric: A calculated metric for this route. How the metric is calculated is implementation-specific. Maximum value (infinity) indicates the route was recently retracted and is temporarily unreachable (see Section 3.5.5 of [I-D.ietf-babel-rfc6126bis]). Either babel-route-calculated-metric or babel-route-received-metric MUST be provided.

babel-route-seqno: The sequence number with which this route was advertised.

babel-route-next-hop: The next-hop address of this route. This will be empty if this route has no next-hop address.

babel-route-feasible: A boolean flag indicating whether this route is feasible, as defined in Section 3.5.1 of [I-D.ietf-babel-rfc6126bis]).

babel-route-selected: A boolean flag indicating whether this route is selected (i.e., whether it is currently being used for forwarding and is being advertised).

4. Common Objects

4.1. Definition of babel-credential-obj

    object {
        credentials          ro babel-cred;
    } babel-credential-obj;

    babel-cred: A credential, such as an X.509 certificate, a public key, etc. used for signing and/or encrypting Babel messages.
4.2. Definition of babel-log-obj

object {
    datetime ro babel-log-time;
    string ro babel-log-entry;
} babel-log-obj;

babel-log-time: The date and time (according to the device internal clock setting, which may be a time relative to boot time, acquired from NTP, configured by the user, etc.) when this log entry was created.

babel-log-entry: The logged message, as a string of utf-8 encoded hex characters.

5. Extending the Information Model

Implementations MAY extend this information model with other parameters or objects. For example, an implementation MAY choose to expose Babel route filtering rules by adding a route filtering object with parameters appropriate to how route filtering is done in that implementation. The precise means used to extend the information model would be specific to the data model the implementation uses to expose this information.

6. Security Considerations

This document defines a set of information model objects and parameters that may be exposed to be visible from other devices, and some of which may be configured. Securing access to and ensuring the integrity of this data is in scope of and the responsibility of any data model derived from this information model. Specifically, any YANG [RFC7950] data model is expected to define security exposure of the various parameters, and a [TR-181] data model will be secured by the mechanisms defined for the management protocol used to transport it.

This information model defines objects that can allow credentials (for this device, for trusted devices, and for trusted certificate authorities) to be added and deleted. Public keys and shared secrets may be exposed through this model. This model requires that private keys never be exposed. The Babel security mechanisms that make use of these credentials (e.g., [I-D.ietf-babel-dtls], [I-D.ietf-babel-hmac]) are expected to define what credentials can be used with those mechanisms.
7. IANA Considerations

This document defines a Babel Link Type registry for the values of the babel-link-type and babel-supported-link-types parameters to be listed under the Babel Routing Protocol registry.

Valid Babel Link Type names are normatively defined as:

- MUST be at least 1 character and no more than 20 characters long
- MUST contain only US-ASCII [RFC0020] letters ‘A’ - ‘Z’ and ‘a’ - ‘z’, digits ‘0’ - ‘9’, and hyphens (‘-’, ASCII 0x2D or decimal 45)
- MUST contain at least one letter (‘A’ - ‘Z’ or ‘a’ - ‘z’)
- MUST NOT begin or end with a hyphen
- hyphens MUST NOT be adjacent to other hyphens

The rules for Link Type names, excepting the limit of 20 characters maximum, are also expressed below (as a non-normative convenience) using ABNF [RFC5234].

```
SRVNAME = *(1*DIGIT [HYPHEN]) ALPHA *([HYPHEN] ALNUM)
ALNUM   = ALPHA / DIGIT ; A-Z, a-z, 0-9
HYPHEN  = %x2D              ; "-"
ALPHA   = %x41-5A / %x61-7A ; A-Z / a-z [RFC5234]
DIGIT   = %x30-39           ; 0-9 [RFC5234]
```

The allocation policy of this registry is Specification Required [RFC8126].

The initial values in the "Babel Link Type" registry are:

```
+----------+-------------------------------------------+------------+
| Name     | Used for Links Defined By                 | Reference  |
+----------+-------------------------------------------+------------+
| ethernet | [IEEE-802.3-2018]                         | (this      |
|          |                                           | document)  |
| other    | to be used when no link type information  | (this      |
|          | available                                 | document)  |
| tunnel   | to be used for a tunneled interface over   | (this      |
|          | unknown physical link                     | document)  |
| wireless | [IEEE-802.11-2016]                        | (this      |
|          |                                           | document)  |
| exp-*    | Reserved for Experimental Use             | (this      |
|          |                                           | document)  |
+----------+-------------------------------------------+------------+
```
8. Acknowledgements

Juliusz Chroboczek, Toke Hoeiland-Joergensen, David Schinazi, Mahesh Jethanandani, Acee Lindem, and Carsten Bormann have been very helpful in refining this information model.

The language in the Notation section was mostly taken from [RFC8193].

9. References

9.1. Normative References

[I-D.ietf-babel-rfc6126bis]


9.2. Informative References

[I-D.ietf-babel-dtls]

[I-D.ietf-babel-hmac]
[IEEE-802.11-2016]
"IEEE Standard 802.11-2016 - IEEE Standard for Information Technology - Telecommunications and information exchange between systems Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications."

[IEEE-802.3-2018]

[ISO.10646]


Appendix A. Open Issues

1. I want to get rid of the security log, because all Babel messages (which should be defined as all messages to/from the udp-port) are be logged by message-log. I don’t like message log as it is. I think if logging is enabled it should just write to a text file. This will mean there also needs to be a means of downloading/reading the log file.

2. Consider the following statistics: under interface object: sent multicast Hello, sent updates, received Babel messages; under neighbor object: sent unicast Hello, sent updates, sent IHU, received Hello, received updates, received IHUs. Would also need to enable/disable stats and clear stats.

3. Security section needs further review

4. Commands to add and delete credentials, and parameters that allow credential to be identified without allowing access to private credential info

5. Check description of enable parameters to make sure ok for YANG and TR-181. Closed by updating description to be useful for YANG and TR-181, using language consistent with YANG descriptions.

6. Distinguish signed and unsigned integers?

7. Review new IANA Considerations section. Should ABNF be normative?

Closed Issues:

1. Datatype of the router-id: Closed by introducing binary datatype and using that for router-id

2. babel-neighbor-address as IPv6-only: Closed by leaving as is (IPv4 and IPv6)

3. babel-implementation-version includes the name of the implementation: Closed by adding "name" to description

4. Delete external-cost?: Closed by deleting.

5. Would it be useful to define some parameters for reporting statistics or logs? [2 logs are now included. If others are needed they need to be proposed. See Open Issues for additional thoughts on logs and statistics.]
6. Closed by defining base64 type and using it for all router IDs: "babel-self-router-id: Should this be an opaque 64-bit value instead of int?"

7. Closed as "No": Do we need a registry for the supported security mechanisms? [Given the current limited set, and unlikelihood of massive expansion, I don’t think so. But we can if someone wants it.]

8. This draft must be reviewed against draft-ietf-babel-rfc6126bis. [I feel like this has been adequately done, but I could be wrong.]

9. babel-interfaces-obj: Juliusz:"This needs further discussion, I fear some of these are implementation details." [In the absence of discussion, the current model stands. Note that all but link-type and the neighbors sub-object are optional. If an implementation does not have any of the optional elements then it simply doesn’t have them and that’s fine.]

10. Would it be useful to define some parameters specifically for security anomalies? [The 2 logs should be useful in identifying security anomalies. If more is needed, someone needs to propose.]

11. I created a basic security model. It’s useful for single (or no) active security mechanism (e.g., just HMAC, just DTLS, or neither); but not multiple active (both HMAC and DTLS -- which is not the same as HMAC of DTLS and would just mean that HMAC would be used on all unencrypted messages -- but right now the model doesn’t allow for configuring HMAC of unencrypted messages for routers without DTLS, while DTLS is used if possible). OK? [No-one said otherwise.]

12. babel-external-cost may need more work. [if no comment, it will be left as is]

13. babel-hello-[mu]cast-history: the Hello history is formatted as 16 bits, per A.1 of 6126bis. Is that a too implementation specific? [We also now have an optional-to-implement log of received messages, and I made these optional. So maybe this is ok?]

14. rxcost, txcost, cost: is it ok to model as integers, since 6126bis 2.1 says costs and metrics need not be integers. [I have them as integers unless someone insists on something else.]
15. For the security log, should it also log whether the credentials were considered ok? [Right now it doesn’t and I think that’s ok because if you log Hellos it was ok and if you don’t it wasn’t.]

16. Should Babel link types have an IANA registry? [Agreed to do this at IETF 102.]

Appendix B. Change Log

Individual Drafts:

v00 2016-07-07 EBD: Initial individual draft version

v01 2017-03-13: Addressed comments received in 2016-07-15 email from J. Chroboczek

Working group drafts:


v01 2018-01-02: Removed item from issue list that was agreed (in Prague) not to be an issue. Added description of data types under Notation section, and used these in all data types. Added babel-security and babel-trust.

v02 2018-04-05:

* changed babel-version description to babel-implementation-version
* replace optional babel-interface-seqno with optional babel-mcast-hello-seqno and babel-ucast-hello-seqno
* replace optional babel-interface-hello-interval with optional babel-mcast-hello-interval and babel-ucast-hello-interval
* remove babel-request-trigger-ack
* remove "babel-router-id: router-id of the neighbor"; note that parameter had previously been removed but description had accidentally not been removed
* added an optional "babel-cost" field to babel-neighbors object, since the spec does not define how exactly the cost is computed from rxcost/txcost
* deleted babel-source-garbage-collection-time

* change babel-lossy-link to babel-link-type and make this an enumeration; added at top level babel-supported-link-types so which are supported by this implementation can be reported

* changes to babel-security-obj to allow self credentials to be one or more instances of a credential object. Allowed trusted credentials to include CA credentials; made some parameter name changes

* updated references and Introduction

* added Overview section

* deleted babel-sources-obj

* added feasible Boolean to routes

* added section to briefly describe extending the information model.

* deleted babel-route-neighbor

* tried to make definition of babel-interface-reference clearer

* added security and message logs

v03 2018-05-31:

* added reference to RFC 8174 (update to RFC 2119 on key words)

* applied edits to Introduction text per Juliusz email of 2018-04-06

* Deleted sentence in definition of "int" data type that said it was also used for enumerations. Changed all enumerations to strings. The only enumerations were for link types, which are now "ethernet", "wireless", "tunnel", and "other".

* deleted [ip-address babel-mcast-group-ipv4;]

* babel-external-cost description changed

* babel-security-self-cred: Added "any private key component of a credential MUST NOT be readable;"
hello-history parameters put recent Hello in most significant bit and length of parameter is not constrained.

* babel-hello-seqno in neighbors-obj changed to babel-exp-mcast-hello-seqno and babel-exp-ucast-hello-seqno

* added babel-route-neighbor back again. It was mistakenly deleted

* changed babel-route-metric and babel-route-announced-metric to babel-route-received-metric and babel-route-calculated-metric

* changed model of security object to put list of supported mechanisms at top level and separate security object per mechanism. This caused some other changes to the security object

v04 2018-10-15:

* changed babel-mcast-group-ipv6 to babel-mcast-group

* link type parameters changed to point to newly defined registry

* babel-ucast-hello-interval moved to neighbor object

* babel-ucast-hello-seqno moved to neighbor object

* babel-neighbor-ihu-interval deleted

* in log descriptions, included statement that there SHOULD be ability to clear logs

* added IANA registry for link types

* added "ro" and "rw" to tables for read-write and read-only

* added metric computation parameter to interface

Author’s Address

Barbara Stark
AT&T
Atlanta, GA
US

Email: barbara.stark@att.com
Babel Information Model

draft-ietf-babel-information-model-05

Abstract

This Babel Information Model can be used to create data models under various data modeling regimes. It allows a Babel implementation (via a management protocol or interface) to report on its current state and may allow some limited configuration of protocol constants.

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 https://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 6, 2019.

Copyright Notice

Copyright (c) 2019 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 (https://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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

Babel is a loop-avoiding distance-vector routing protocol defined in [I-D.ietf-babel-rfc6126bis]. [I-D.ietf-babel-hmac] defines a security mechanism that allows Babel packets to be cryptographically authenticated, and [I-D.ietf-babel-dtls] defines a security mechanism that allows Babel packets to be encrypted. This document describes an information model for Babel (including implementations using one of these security mechanisms) that can be used to create management protocol data models (such as a NETCONF [RFC6241] YANG [RFC7950] data model).

Due to the simplicity of the Babel protocol, most of the information model is focused on reporting Babel protocol operational state, and very little of that is considered mandatory to implement (contingent on a management protocol with Babel support being implemented). Some parameters may be configurable. However, it is up to the Babel implementation whether to allow any of these to be configured within its implementation. Where the implementation does not allow...
configuration of these parameters, it may still choose to expose them as read-only.

The Information Model is presented using a hierarchical structure. This does not preclude a data model based on this Information Model from using a referential or other structure.

1.1. Requirements Language

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] and updated by [RFC8174].

1.2. Notation

This document uses a programming language-like notation to define the properties of the objects of the information model. An optional property is enclosed by square brackets, [ ], and a list property is indicated by two numbers in angle brackets, <m..n>, where m indicates the minimal number of list elements, and n indicates the maximum number of list elements. The symbol * for n means there are no defined limits on the number of list elements. Each parameter and object includes an indication of "ro" or "rw". "ro" means the parameter or object is read-only. "rw" means it is read-write. For an object, read-write means instances of the object can be created or deleted. If an implementation is allowed to choose to implement a "rw" parameter as read-only, this is noted in the parameter description.

The object definitions use base types that are defined as follows:

- **binary**: A binary string (sequence of octets).
- **boolean**: A type representing a Boolean value.
- **counter**: A non-negative integer that monotonically increases. Counters may have discontinuities and they are not expected to persist across restarts.
- **datetime**: A type representing a date and time using the Gregorian calendar. The datetime format MUST conform to RFC 3339 [RFC3339].
- **ip-address**: A type representing an IP address. This type supports both IPv4 and IPv6 addresses.
operation  A type representing a remote procedure call or other action that can be used to manipulate data elements or system behaviors.

reference  A type representing a reference to another information or data model element or to some other device resource.

string    A type representing a human-readable string consisting of a (possibly restricted) subset of Unicode and ISO/IEC 10646 [ISO.10646] characters.

uint      A type representing an unsigned integer number. This information model does not define a precision.

2.  Overview

The Information Model is hierarchically structured as follows:

  +++ babel-information
  |  +++ babel-implementation-version
  |  +++ babel-enable
  |  +++ router-id
  |  +++ babel-supported-link-types
  |  +++ self-seqno
  |  +++ babel-metric-comp-algorithms
  |  +++ babel-security-supported
  |  +++ babel-hmac-enable
  |  +++ babel-hmac-algorithms
  |  +++ babel-dtls-enable
  |  +++ babel-dtls-cert-types
  |  +++ babel-stats-enable
  |  +++ babel-stats-reset
  |  +++ bible-constants
  |  |  +++ bible-udp-port
  |  |  +++ bible-mcast-group
  |  +++ bible-interfaces
  |  |  +++ bible-interface-reference
  |  |  +++ bible-interface-enable
  |  |  +++ bible-link-type
  |  |  +++ bible-interface-metric-algorithm
  |  |  +++ bible-mcast-hello-seqno
  |  |  +++ bible-mcast-hello-interval
  |  |  +++ bible-update-interval
  |  |  +++ bible-packet-log-enable
  |  |  +++ bible-packet-log
  |  |  +++ bible-if-stats
  |  |  |  +++ bible-sent-mcast-hello
  |  |  |  +++ bible-sent-mcast-update
--- babel-received-packets
  +-- babel-neighbor-address
  +-- babel-hello-mcast-history
  +-- babel-hello-ucast-history
  +-- babel-txcost
  +-- babel-exp-mcast-hello-seqno
  +-- babel-exp-ucast-hello-seqno
  +-- babel-ucast-hello-seqno
  +-- babel-ucast-hello-interval
  +-- babel-rxcost
  +-- babel-cost
  +-- babel-nbr-stats
    +-- babel-sent-ucast-hello
    +-- babel-sent-ucast-update
    +-- babel-sent-IHU
    +-- babel-received-hello
    +-- babel-received-update
    +-- babel-received-IHU

--- babel-routes
  +-- babel-route-prefix
  +-- babel-route-prefix-length
  +-- babel-route-router-id
  +-- babel-route-neighbor
  +-- babel-route-received-metric
  +-- babel-route-calculated-metric
  +-- babel-route-seqno
  +-- babel-route-next-hop
  +-- babel-route-feasible
  +-- babel-route-selected

--- babel-hmac
  +-- babel-hmac-algorithm
  +-- babel-hmac-verify
  +-- babel-hmac-interfaces
    +-- babel-hmac-key-name
    +-- babel-hmac-key-use-sign
    +-- babel-hmac-key-use-verify
    +-- babel-hmac-key-value

--- babel-dtls
  +-- babel-dtls-interfaces
  +-- babel-dtls-cached-info
  +-- babel-dtls-cert-prefer
    +-- babel-cert-value
    +-- babel-cert-type
    +-- babel-cert-private-key
    +-- babel-cert-test
Most parameters are read-only. Following is a descriptive list of the parameters that are not required to be read-only:

- enable/disable Babel
- babel-hmac objects
- babel-dtls objects
- enable/disable statistics collection
- Constant: UDP port
- Constant: IPv6 multicast group
- Interface: Link type
- Interface: External cost (must be configurable if implemented, but implementation is optional)
- Interface: enable/disable Babel on this interface
- Interface: enable/disable packet log
- HMAC: algorithm
- HMAC: verify received packets
- HMAC: interfaces
- HMAC-keys: create new entries
- HMAC-keys: use to sign packets
- HMAC-keys: use to verify packets
- DTLS: interfaces
- DTLS: use cached info extensions
- DTLS: preferred order of certificate types
- DTLS-certs: create new entries

The following parameters are required to return no value when read:

- HMAC key values
3. The Information Model

3.1. Definition of babel-information-obj

object {
  string               ro babel-implementation-version;
  boolean              rw babel-enable;
  binary               ro babel-self-router-id;
  [uint ro babel-self-seqno;]
  string               ro babel-metric-comp-algorithms<1..*>;
  string               ro babel-security-supported<0..*>;
  [boolean ro babel-hmac-enable;]
  [string ro babel-hmac-algorithms<1..*>;]
  [boolean ro babel-dtls-enable;]
  [string ro babel-dtls-cert-types<1..*>;]
  [boolean rw babel-stats-enable;]
  [operation               babel-stats-reset;]
  babel-constants-obj  ro babel-constants;
  babel-interfaces-obj ro babel-interfaces<0..*>;
  babel-routes-obj     ro babel-routes<0..*>;
  [babel-hmac-obj       rw babel-hmac<0..*>;]
  [babel-dtls-obj       rw babel-dtls<0..*>;]
} babel-information-obj;

babel-implementation-version: The name and version of this implementation of the Babel protocol.

babel-enable: When written, it configures whether the protocol should be enabled (true) or disabled (false). A read from the running or intended datastore indicates the configured administrative value of whether the protocol is enabled (true) or not (false). A read from the operational datastore indicates whether the protocol is actually running (true) or not (i.e., it indicates the operational state of the protocol). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").
babel-self-router-id: The router-id used by this instance of the Babel protocol to identify itself. [I-D.ietf-babel-rfc6126bis] describes this as an arbitrary string of 8 octets.

babel-supported-link-types: Lists the set of link types supported by this instance of Babel. Valid enumeration values are defined in the Babel Link Types registry (see Section 6).

babel-self-seqno: The current sequence number included in route updates for routes originated by this node. This is a 16-bit unsigned integer.

babel-metric-comp-algorithms: List of supported cost computation algorithms. Possible values include "k-out-of-j", and "ETX".

babel-security-supported: List of supported security mechanisms. Possible values include "HMAC" and "DTLS".

babel-hmac-enable: Indicates whether the HMAC security mechanism is enabled (true) or disabled (false). An implementation MAY choose to expose this parameter as read-only ("ro").

babel-hmac-algorithms: List of supported HMAC computation algorithms. Possible values include "HMAC-SHA256", "BLAKE2s".

babel-dtls-enable: Indicates whether the DTLS security mechanism is enabled (true) or disabled (false). An implementation MAY choose to expose this parameter as read-only ("ro").

babel-dtls-cert-types: List of supported DTLS certificate types. Possible values include "X.509" and "RawPublicKey".

babel-stats-enable: Indicates whether statistics collection is enabled (true) or disabled (false) on all interfaces, including neighbor-specific statistics (babel-nbr-stats).

babel-stats-reset: An operation that resets all babel-if-stats and babel-nbr-stats parameters to zero. This operation has no input or output parameters.

babel-constants: A babel-constants-obj object.

babel-interfaces: A set of babel-interface-obj objects.

babel-routes: A set of babel-route-obj objects. Contains the routes known to this node.
babel-hmac: A babel-hmac-obj object. If this object is implemented, it provides access to parameters related to the HMAC security mechanism. An implementation MAY choose to expose this object as read-only ("ro").

babel-dtls: A babel-dtls-obj object. If this object is implemented, it provides access to parameters related to the DTLS security mechanism. An implementation MAY choose to expose this object as read-only ("ro").

3.2. Definition of babel-constants-obj

object {
    uint         rw babel-udp-port;
    [ip-address   rw babel-mcast-group;]
} babel-constants-obj;

babel-udp-port: UDP port for sending and listening for Babel packets. Default is 6696. An implementation MAY choose to expose this parameter as read-only ("ro"). This is a 16-bit unsigned integer.

babel-mcast-group: Multicast group for sending and listening to multicast announcements on IPv6. Default is ff02:0:0:0:0:0:1:6. An implementation MAY choose to expose this parameter as read-only ("ro").

3.3. Definition of babel-interfaces-obj

object {
    reference            ro babel-interface-reference;
    [boolean              rw babel-interface-enable;]
    string               rw babel-link-type;
    string               ro babel-interface-metric-algorithm;
    [uint                 ro babel-mcast-hello-seqno;]
    [uint                 ro babel-mcast-hello-interval;]
    [uint                 ro babel-update-interval;]
    [boolean              rw babel-packet-log-enable;]
    [reference            ro babel-packet-log;]
    [babel-if-stats-obj   ro babel-if-stats;]
    [babel-neighbors-obj  ro babel-neighbors<0..*>;]
} babel-interfaces-obj;

babel-interface-reference: Reference to an interface object as defined by the data model (e.g., YANG [RFC7950], BBF [TR-181]). Data model is assumed to allow for referencing of interface objects which may be at any layer (physical, Ethernet MAC, IP, tunneled IP, etc.). Referencing syntax will be specific to the
data model. If there is no set of interface objects available, this should be a string that indicates the interface name used by the underlying operating system.

**babel-interface-enable:** When written, it configures whether the protocol should be enabled (true) or disabled (false) on this interface. A read from the running or intended datastore indicates the configured administrative value of whether the protocol is enabled (true) or not (false). A read from the operational datastore indicates whether the protocol is actually running (true) or not (i.e., it indicates the operational state of the protocol). A data model that does not replicate parameters for running and operational datastores can implement this as two separate parameters. An implementation MAY choose to expose this parameter as read-only ("ro").

**babel-link-type:** Indicates the type of link. The value MUST be one of those listed in the babel-supported-link-types parameter. Valid enumeration values are identified in Babel Link Types registry. An implementation MAY choose to expose this parameter as read-only ("ro").

**babel-interface-metric-algorithm:** Indicates the metric computation algorithm used on this interface. The value MUST be one of those listed in the babel-information-obj babel-metric-comp-algorithms parameter.

**babel-mcast-hello-seqno:** The current sequence number in use for multicast Hellos sent on this interface. This is a 16-bit unsigned integer.

**babel-mcast-hello-interval:** The current interval in use for multicast Hellos sent on this interface. Units are centiseconds. This is a 16-bit unsigned integer.

**babel-update-interval:** The current interval in use for all updates (multicast and unicast) sent on this interface. Units are centiseconds. This is a 16-bit unsigned integer.

**babel-packet-log-enable:** Indicates whether packet logging is enabled (true) or disabled (false) on this interface.

**babel-packet-log:** A reference or url link to a file that contains a timestamped log of packets received and sent on babel-udp-port on this interface. The [libpcap] file format with .pcap file extension SHOULD be supported for packet log files. Logging is enabled / disabled by babel-packet-log-enable.
babel-if-stats: Statistics collection object for this interface.

babel-neighbors: A set of babel-neighbors-obj objects.

3.4. Definition of babel-if-stats-obj

    object {
      uint                    ro babel-sent-mcast-hello;
      uint                    ro babel-sent-mcast-update;
      uint                    ro babel-received-packets;
    } babel-if-stats-obj;

babel-sent-mcast-hello: A count of the number of multicast Hello packets sent on this interface.

babel-sent-mcast-update: A count of the number of multicast update packets sent on this interface.

babel-received-packets: A count of the number of Babel packets received on this interface.

3.5. Definition of babel-neighbors-obj

    object {
      ip-address               ro babel-neighbor-address;
      [binary                   ro babel-hello-mcast-history;]
      [binary                   ro babel-hello-ucast-history;]
      uint                     ro babel-txcost;
      uint                     ro babel-exp-mcast-hello-seqno;
      uint                     ro babel-exp-ucast-hello-seqno;
      [uint                     ro babel-ucast-hello-seqno;]
      [uint                     ro babel-ucast-hello-interval;]
      [uint                     ro babel-rxcost;]
      [uint                     ro babel-cost;]
      [babel-nbr-stats-obj      ro babel-nbr-stats;]
    } babel-neighbors-obj;

babel-neighbor-address: IPv4 or IPv6 address the neighbor sends packets from.

babel-hello-mcast-history: The multicast Hello history of whether or not the multicast Hello packets prior to babel-exp-mcast-hello-seqno were received. A binary sequence where the most recently received Hello is expressed as a "1" placed in the left-most bit, with prior bits shifted right (and "0" bits placed between prior Hello bits and most recent Hello for any not-received Hellos). This value should be displayed using hex digits ([0-9a-fA-F]). See [I-D.ietf-babel-rfc6126bis], section A.1.
babel-hello-ucast-history: The unicast Hello history of whether or not the unicast Hello packets prior to babel-exp-ucast-hello-seqno were received. A binary sequence where the most recently received Hello is expressed as a "1" placed in the left-most bit, with prior bits shifted right (and "0" bits placed between prior Hello bits and most recent Hello for any not-received Hellos). This value should be displayed using hex digits ([0-9a-fA-F]). See [I-D.ietf-babel-rfc6126bis], section A.1.

babel-txcost: Transmission cost value from the last IHU packet received from this neighbor, or maximum value to indicate the IHU hold timer for this neighbor has expired. See [I-D.ietf-babel-rfc6126bis], section 3.4.2. This is a 16-bit unsigned integer.

babel-exp-mcast-hello-seqno: Expected multicast Hello sequence number of next Hello to be received from this neighbor. If multicast Hello packets are not expected, or processing of multicast packets is not enabled, this MUST be 0. This is a 16-bit unsigned integer.

babel-exp-ucast-hello-seqno: Expected unicast Hello sequence number of next Hello to be received from this neighbor. If unicast Hello packets are not expected, or processing of unicast packets is not enabled, this MUST be 0. This is a 16-bit unsigned integer.

babel-ucast-hello-seqno: The current sequence number in use for unicast hellos sent to this neighbor. This is a 16-bit unsigned integer.

babel-ucast-hello-interval: The current interval in use for unicast hellos sent to this neighbor. Units are centiseconds. This is a 16-bit unsigned integer.

babel-rxcost: Reception cost calculated for this neighbor. This value is usually derived from the Hello history, which may be combined with other data, such as statistics maintained by the link layer. The rxcost is sent to a neighbor in each IHU. See [I-D.ietf-babel-rfc6126bis], section 3.4.3. This is a 16-bit unsigned integer.

babel-cost: Link cost is computed from the values maintained in the neighbor table: the statistics kept in the neighbor table about the reception of Hellos, and the txcost computed from received IHU packets. This is a 16-bit unsigned integer.

babel-nbr-stats: Statistics collection object for this neighbor.
3.6. Definition of babel-nbr-stats-obj

object {
  uint                 ro babel-sent-ucast-hello;
  uint                 ro babel-sent-ucast-update;
  uint                 ro babel-sent-IHU;
  uint                 ro babel-received-hello;
  uint                 ro babel-received-update;
  uint                 ro babel-received-IHU;
} babel-nbr-stats-obj;

babel-sent-ucast-hello: A count of the number of unicast Hello packets sent to this neighbor.

babel-sent-ucast-update: A count of the number of unicast update packets sent to this neighbor.

babel-sent-IHU: A count of the number of IHU packets sent to this neighbor.

babel-received-hello: A count of the number of Hello packets received from this neighbor.

babel-received-update: A count of the number of update packets received from this neighbor.

babel-received-IHU: A count of the number of IHU packets received from this neighbor.

3.7. Definition of babel-routes-obj

object {
  ip-address           ro babel-route-prefix;
  uint                 ro babel-route-prefix-length;
  binary               ro babel-route-router-id;
  string               ro babel-route-neighbor;
  [uint                 ro babel-route-received-metric;]
  [uint                 ro babel-route-calculated-metric;]
  uint                 ro babel-route-seqno;
  ip-address           ro babel-route-next-hop;
  boolean              ro babel-route-feasible;
  boolean              ro babel-route-selected;
} babel-routes-obj;

babel-route-prefix: Prefix (expressed in IP address format) for which this route is advertised.
babel-route-prefix-length: Length of the prefix for which this route is advertised.

babel-route-router-id: router-id of the source router for which this route is advertised.

babel-route-neighbor: Reference to the babel-neighbors entry for the neighbor that advertised this route.

babel-route-received-metric: The metric with which this route was advertised by the neighbor, or maximum value to indicate the route was recently retracted and is temporarily unreachable (see Section 3.5.5 of [I-D.ietf-babel-rfc6126bis]). This metric will be 0 (zero) if the route was not received from a neighbor but was generated through other means. Either babel-route-calculated-metric or babel-route-received-metric MUST be provided. This is a 16-bit unsigned integer.

babel-route-calculated-metric: A calculated metric for this route. How the metric is calculated is implementation-specific. Maximum value indicates the route was recently retracted and is temporarily unreachable (see Section 3.5.5 of [I-D.ietf-babel-rfc6126bis]). Either babel-route-calculated-metric or babel-route-received-metric MUST be provided. This is a 16-bit unsigned integer.

babel-route-seqno: The sequence number with which this route was advertised. This is a 16-bit unsigned integer.

babel-route-next-hop: The next-hop address of this route. This will be empty if this route has no next-hop address.

babel-route-feasible: A Boolean flag indicating whether this route is feasible, as defined in Section 3.5.1 of [I-D.ietf-babel-rfc6126bis]).

babel-route-selected: A Boolean flag indicating whether this route is selected (i.e., whether it is currently being used for forwarding and is being advertised).

3.8. Definition of babel-hmac-obj
object {
    string   rw babel-hmac-algorithm;
    boolean  rw babel-hmac-verify;
    boolean  rw babel-hmac-apply-all;
    reference rw babel-hmac-interfaces<0..*>;
    babel-hmac-keys-obj   rw babel-hmac-keys<0..*>;
} babel-hmac-obj;

babel-hmac-algorithm The name of the HMAC algorithm this object instance uses. The value MUST be the same as one of the enumerations listed in the babel-hmac-algorithms parameter. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-hmac-verify A Boolean flag indicating whether HMAC hashes in incoming Babel packets are required to be present and are verified. If this parameter is "true", incoming packets are required to have a valid HMAC hash. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-hmac-apply-all: A Boolean flag indicating whether this babel-hmac instance is to be used for all interfaces. If "true", this instance applies to all interfaces and the babel-hmac-interfaces parameter is ignored. If babel-hmac-apply-all is "true", there MUST NOT be other instances of the babel-hmac object. If "false", the babel-hmac-interfaces parameter determines which interfaces this instance applies to. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-hmac-interfaces: List of references to the babel-interfaces entries this babel-hmac entry applies to. This parameter is ignored if babel-hmac-apply-all is "true". An interface MUST NOT be listed in multiple instances of the babel-hmac object. An implementation MAY choose to expose this parameter as read-only ("ro").


3.9. Definition of babel-hmac-keys-obj

object {
    string       ro babel-hmac-key-name;
    boolean      rw babel-hmac-key-use-sign;
    boolean      rw babel-hmac-key-use-verify;
    binary       -- babel-hmac-key-value;
    {operation    babel-hmac-key-test;}
} babel-hmac-keys-obj;
babel-hmac-key-name: A unique name for this HMAC key that can be used to identify the key in this object instance, since the key value is not allowed to be read. This value can only be provided when this instance is created, and is not subsequently writable.

babel-key-use-sign: Indicates whether this key value is used to sign sent Babel packets. Sent packets are signed using this key if the value is "true". If the value is "false", this key is not used to sign sent Babel packets. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-key-use-verify: Indicates whether this key value is used to verify incoming Babel packets. This key is used to verify incoming packets if the value is "true". If the value is "false", no HMAC is computed from this key for comparing an incoming packet. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-key-value: The value of the HMAC key. An implementation MUST NOT allow this parameter to be read. This can be done by always providing an empty string, or through permissions, or other means. This value can only be provided when this instance is created, and is not subsequently writable.

babel-hmac-test: An operation that allows the HMAC key and hash algorithm to be tested to see if they produce an expected outcome. Input to this operation is a binary string. The implementation is expected to create a hash of this string using the babel-hmac-key-value and the babel-hmac-algorithm. The output of this operation is the resulting hash, as a binary string.

3.10. Definition of babel-dtls-obj

object {
    boolean     rw babel-dtls-apply-all;
    reference   rw babel-dtls-interfaces<0..*>;
    [boolean    rw babel-dtls-cached-info];
    [string     rw babel-dtls-cert-prefer<0..*>];
    babel-dtls-certs-obj rw babel-dtls-certs<0..*>;
} babel-dtls-obj;

babel-dtls-apply-all: A Boolean flag indicating whether this babel-dtls instance is to be used for all interfaces. If "true", this instance applies to all interfaces and the babel-dtls-interfaces parameter is ignored. If babel-dtls-apply-all is "true", there MUST NOT be other instances of the babel-dtls object. If "false", the babel-dtls-interfaces parameter determines which interfaces
this instance applies to. An implementation MAY choose to expose
this parameter as read-only ("ro").

babel-dtls-interfaces: List of references to the babel-interfaces
entries this babel-dtls entry applies to. This parameter is
ignored if babel-dtls-apply-all is "true". An interface MUST NOT
be listed in multiple instances of the babel-dtls object. If this
list is empty, then it applies to all interfaces. An
implementation MAY choose to expose this parameter as read-only
("ro").

babel-dtls-cached-info: Indicates whether the cached_info extension
is included in ClientHello and ServerHello packets. The extension
is included if the value is "true". An implementation MAY choose
to expose this parameter as read-only ("ro").

babel-dtls-cert-prefer: List of supported certificate types, in
order of preference. The values MUST be among those listed in the
babel-dtls-cert-types parameter. This list is used to populate
the server_certificate_type extension in a Client Hello. Values
that are present in at least one instance in the babel-dtls-certs
object with a non-empty babel-cert-private-key will be used to
populate the client_certificate_type extension in a Client Hello.

babel-dtls-certs: A set of babel-dtls-keys-obj objects. This
contains both certificates for this implementation to present for
authentication, and to accept from others. Certificates with a
non-empty babel-cert-private-key can be presented by this
implementation for authentication.

3.11. Definition of babel-dtls-certs-obj

object {
  string              ro babel-cert-value;
  string              ro babel-cert-type;
  binary              -- babel-cert-private-key;
  [operation          babel-cert-test;]
} babel-dtls-certs-obj;

babel-cert-value: The DTLS certificate in PEM format [RFC7468].
This value can only be provided when this instance is created, and
is not subsequently writable.

babel-cert-type: The name of the certificate type of this object
instance. The value MUST be the same as one of the enumerations
listed in the babel-dtls-cert-types parameter. This value can
only be provided when this instance is created, and is not
subsequently writable.
babel-cert-private-key: The value of the private key. If this is non-empty, this certificate can be used by this implementation to provide a certificate during DTLS handshaking. An implementation MUST NOT allow this parameter to be read. This can be done by always providing an empty string, or through permissions, or other means. This value can only be provided when this instance is created, and is not subsequently writable.

babel-cert-test: An operation that allows a hash of the provided input string to be created using the certificate public key and the SHA-256 hash algorithm. Input to this operation is a binary string. The output of this operation is the resulting hash, as a binary string.

4. Extending the Information Model

Implementations MAY extend this information model with other parameters or objects. For example, an implementation MAY choose to expose Babel route filtering rules by adding a route filtering object with parameters appropriate to how route filtering is done in that implementation. The precise means used to extend the information model would be specific to the data model the implementation uses to expose this information.

5. Security Considerations

This document defines a set of information model objects and parameters that may be exposed to be visible from other devices, and some of which may be configured. Securing access to and ensuring the integrity of this data is in scope of and the responsibility of any data model derived from this information model. Specifically, any YANG [RFC7950] data model is expected to define security exposure of the various parameters, and a [TR-181] data model will be secured by the mechanisms defined for the management protocol used to transport it.

This information model defines objects that can allow credentials (for this device, for trusted devices, and for trusted certificate authorities) to be added and deleted. Public keys and shared secrets may be exposed through this model. This model requires that private keys never be exposed. The Babel security mechanisms that make use of these credentials (e.g., [I-D.ietf-babel-dtls], [I-D.ietf-babel-hmac]) are expected to define what credentials can be used with those mechanisms.
6. IANA Considerations

This document defines a Babel Link Type registry for the values of the babel-link-type and babel-supported-link-types parameters to be listed under the Babel Routing Protocol registry.

Valid Babel Link Type names are normatively defined as:

- MUST be at least 1 character and no more than 20 characters long
- MUST contain only US-ASCII [RFC0020] letters ‘A’ - ‘Z’ and ‘a’ - ‘z’, digits ‘0’ - ‘9’, and hyphens (’-’, ASCII 0x2D or decimal 45)
- MUST contain at least one letter (‘A’ - ‘Z’ or ‘a’ - ‘z’)
- MUST NOT begin or end with a hyphen
- hyphens MUST NOT be adjacent to other hyphens

The rules for Link Type names, excepting the limit of 20 characters maximum, are also expressed below (as a non-normative convenience) using ABNF [RFC5234].

```
SRVNAME = *(1*DIGIT [HYPHEN]) ALPHA *([HYPHEN] ALNUM)
ALNUM  = ALPHA / DIGIT ; A-Z, a-z, 0-9
HYPHEN = %x2D              ; "-"
ALPHA  = %x41-5A / %x61-7A ; A-Z / a-z [RFC5234]
DIGIT  = %x30-39           ; 0-9      [RFC5234]
```

The allocation policy of this registry is Specification Required [RFC8126].

The initial values in the "Babel Link Type" registry are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Used for Links Defined By</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethernet</td>
<td>[IEEE-802.3-2018]</td>
<td>(this document)</td>
</tr>
<tr>
<td>other</td>
<td>to be used when no link type information available</td>
<td>(this document)</td>
</tr>
<tr>
<td>tunnel</td>
<td>to be used for a tunneled interface over unknown physical link</td>
<td>(this document)</td>
</tr>
<tr>
<td>wireless</td>
<td>[IEEE-802.11-2016]</td>
<td>(this document)</td>
</tr>
<tr>
<td>exp-*</td>
<td>Reserved for Experimental Use</td>
<td>(this document)</td>
</tr>
</tbody>
</table>
7. Acknowledgements

Juliusz Chroboczek, Toke Hoeiland-Joergensen, David Schinazi, Acee Lindem, and Carsten Bormann have been very helpful in refining this information model.

The language in the Notation section was mostly taken from [RFC8193].

8. References

8.1. Normative References

[I-D.ietf-babel-rfc6126bis]


8.2. Informative References
[I-D.ietf-babel-dtls]

[I-D.ietf-babel-hmac]

[IEEE-802.11-2016]

[IEEE-802.3-2018]

[ISO.10646]


Appendix A. Open Issues

All open issues have been closed.

Closed Issues:

1. HMAC spec adds other parameters to neighbor table. Check these to see if any need to be readable or writable. / None were identified.

2. Actions to add and delete HMAC and DTLS credentials, and parameters that allow credential to be identified without allowing access to private credential info. Will have separate sub-tables for HMAC and DTLS credentials. / Instead, there is a normative statement that the parameter values must never be supplied when read.

3. Consider the following statistics: under interface object: sent multicast Hello, sent updates, received Babel messages; under neighbor object: sent unicast Hello, sent updates, sent IHU, received Hello, received updates, received IHUs. Would also need to enable/disable stats and clear stats.

4. Message log (optional to implement) is still in. Support for the libpcap file format is "SHOULD".

5. Single security table with (optional) reference to interfaces that security mechanism applies to. / This actually became separate objects for DTLS and HMAC.

6. Should ABNF be normative in IANA Considerations section? Decision was to leave it as is.

7. I want to get rid of the security log, because all Babel messages (which should be defined as all messages to/from the
udp-port) are be logged by message-log. I don’t like message log as it is. I think if logging is enabled it should just write to a text file. This will mean there also needs to be a means of downloading/reading the log file. Closed by having single log for all messages to/from udp port and log is represented by a string that can be reference to filename or some other part of the overall data model (depends on data model).

8. Check description of enable parameters to make sure ok for YANG and TR-181. Closed by updating description to be useful for YANG and TR-181, using language consistent with YANG descriptions. Done.

9. Distinguish signed and unsigned integers? All integers are unsigned and size is mentioned in description of each uint parameter.

10. Datatype of the router-id: Closed by introducing binary datatype and using that for router-id

11. babel-neighbor-address as IPv6-only: Closed by leaving as is (IPv4 and IPv6)

12. babel-implementation-version includes the name of the implementation: Closed by adding "name" to description


14. Would it be useful to define some parameters for reporting statistics or logs? [2 logs are now included. If others are needed they need to be proposed. See Open Issues for additional thoughts on logs and statistics.]

15. Closed by defining base64 type and using it for all router IDs: "babel-self-router-id: Should this be an opaque 64-bit value instead of int?"

16. Closed as "No": Do we need a registry for the supported security mechanisms? [Given the current limited set, and unlikelihood of massive expansion, I don’t think so. But we can if someone wants it.]

17. This draft must be reviewed against draft-ietf-babel-rfc6126bis. [I feel like this has been adequately done, but I could be wrong.]
18. babel-interfaces-obj: Juliusz: "This needs further discussion, I fear some of these are implementation details." [In the absence of discussion, the current model stands. Note that all but link-type and the neighbors sub-object are optional. If an implementation does not have any of the optional elements then it simply doesn’t have them and that’s fine.]

19. Would it be useful to define some parameters specifically for security anomalies? [The 2 logs should be useful in identifying security anomalies. If more is needed, someone needs to propose.]

20. I created a basic security model. It’s useful for single (or no) active security mechanism (e.g., just HMAC, just DTLS, or neither); but not multiple active (both HMAC and DTLS -- which is not the same as HMAC of DTLS and would just mean that HMAC would be used on all unencrypted messages -- but right now the model doesn’t allow for configuring HMAC of unencrypted messages for routers without DTLS, while DTLS is used if possible). OK? [No-one said otherwise.]

21. babel-external-cost may need more work. [if no comment, it will be left as is]

22. babel-hello-[mu]cast-history: the Hello history is formatted as 16 bits, per A.1 of 6126bis. Is that a too implementation specific? [We also now have an optional-to-implement log of received messages, and I made these optional. So maybe this is ok?]

23. rxcost, txcost, cost: is it ok to model as integers, since 6126bis 2.1 says costs and metrics need not be integers. [I have them as integers unless someone insists on something else.]

24. For the security log, should it also log whether the credentials were considered ok? [Right now it doesn’t and I think that’s ok because if you log Hellos it was ok and if you don’t it wasn’t.]

25. Should Babel link types have an IANA registry? [Agreed to do this at IETF 102.]

Appendix B. Change Log

Individual Drafts:

v00 2016-07-07 EBD: Initial individual draft version
v01 2017-03-13: Addressed comments received in 2016-07-15 email from J. Chroboczek

Working group drafts:


v01 2018-01-02: Removed item from issue list that was agreed (in Prague) not to be an issue. Added description of data types under Notation section, and used these in all data types. Added babel-security and babel-trust.

v02 2018-04-05:

* changed babel-version description to babel-implementation-version
* replace optional babel-interface-segno with optional babel-mcast-hello-segno and babel-ucast-hello-segno
* replace optional babel-interface-hello-interval with optional babel-mcast-hello-interval and babel-ucast-hello-interval
* remove babel-request-trigger-ack
* remove "babel-router-id: router-id of the neighbor"; note that parameter had previously been removed but description had accidentally not been removed
* added an optional "babel-cost" field to babel-neighbors object, since the spec does not define how exactly the cost is computed from rxcost/txcost
* deleted babel-source-garbage-collection-time
* change babel-lossy-link to babel-link-type and make this an enumeration; added at top level babel-supported-link-types so which are supported by this implementation can be reported
* changes to babel-security-obj to allow self credentials to be one or more instances of a credential object. Allowed trusted credentials to include CA credentials; made some parameter name changes
* updated references and Introduction
* added Overview section
* deleted babel-sources-obj
* added feasible Boolean to routes
* added section to briefly describe extending the information model.
* deleted babel-route-neighbor
* tried to make definition of babel-interface-reference clearer
* added security and message logs

v03 2018-05-31:

* added reference to RFC 8174 (update to RFC 2119 on key words)
* applied edits to Introduction text per Juliusz email of 2018-04-06
* Deleted sentence in definition of "int" data type that said it was also used for enumerations. Changed all enumerations to strings. The only enumerations were for link types, which are now "ethernet", "wireless", "tunnel", and "other".
* deleted [ip-address babel-mcast-group-ipv4;]
* babel-external-cost description changed
* babel-security-self-cred: Added "any private key component of a credential MUST NOT be readable;"
* hello-history parameters put recent Hello in most significant bit and length of parameter is not constrained.
* babel-hello-seqno in neighbors-obj changed to babel-exp-mcast-hello-seqno and babel-exp-ucast-hello-seqno
* added babel-route-neighbor back again. It was mistakenly deleted
* changed babel-route-metric and babel-route-announced-metric to babel-route-received-metric and babel-route-calculated-metric
* changed model of security object to put list of supported mechanisms at top level and separate security object per
mechanism. This caused some other changes to the security object

v04 2018-10-15:
* changed babel-mcast-group-ipv6 to babel-mcast-group
* link type parameters changed to point to newly defined registry
* babel-ucast-hello-interval moved to neighbor object
* babel-ucast-hello-seqno moved to neighbor object
* babel-neighbor-ihu-interval deleted
* in log descriptions, included statement that there SHOULD be ability to clear logs
* added IANA registry for link types
* added "ro" and "rw" to tables for read-write and read-only
* added metric computation parameter to interface

v05 2019-01-15:
* security modeled with single table under information-obj and references to interfaces that instance applies to
* changed int to uint because all integers in model were unsigned; added size of integer to description of each uint parameter
* deleted log object and made single message log that points to file or other data model object used to maintain logs
* deleted babel-credentials; there are no more "common" objects; hmac keys and DTLS certificates are more explicitly modeled
* changed definition of babel-security-supported
* added parameters for HMAC and DTLS
* added statistics
* changed all instances of "message" to "packet"
Authors’ Addresses

Barbara Stark
AT&T
Atlanta, GA
US

Email: barbara.stark@att.com

Mahesh Jethanandani
VMware
California
US

Email: mjethanandani@gmail.com
Abstract

This document defines a data model for the Babel routing protocol. The data model is defined using the YANG data modeling language.

Requirements Language

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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here..

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 https://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 6, 2019.

Copyright Notice

Copyright (c) 2019 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 (https://trustee.ietf.org/license-info) in effect on the date of
1. Introduction

This document defines a data model for the Babel routing protocol [I-D.ietf-babel-rfc6126bis]. The data model is defined using YANG 1.1 [RFC7950] data modeling language and is Network Management Datastore Architecture (NDMA) [RFC8342] compatible. It is based on the Babel Information Model [I-D.ietf-babel-information-model].

1.1. Note to RFC Editor

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements and remove this note before publication.

- "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement.
- "ZZZZ" --> the assigned RFC value for Babel Information Model [I-D.ietf-babel-information-model]
1.2. Definitions and Acronyms

1.3. Tree Diagram

For a reference to the annotations used in tree diagrams included in this draft, please see YANG Tree Diagrams [RFC8340].

2. Babel Module

This document defines a YANG 1.1 [RFC7950] data model for the configuration and management of Babel. The YANG module is based on the Babel Information Model [I-D.ietf-babel-information-model].

2.1. Information Model

The following diagram illustrates a top level hierarchy of the model. In addition to information like the version number implemented by this device, the model contains subtrees on constants, interfaces, routes and security.
module: ietf-babel
augment /rt:routing/rt:control-plane-protocols/
rt:control-plane-protocol:
  +++-rw babel!
  +++-ro version? string
  +++-rw enable? boolean
  +++-ro router-id binary
  +++-rw link-type* identityref
  +++-ro sequence-number? uint16
  +++-rw metric-comp-algorithms* identityref
  +++-rw security-supported* identityref
  +++-rw hmac-enable? boolean
  +++-rw hmac-algorithms* identityref
  +++-rw dtls-enable? boolean
  +++-rw dtls-cert-types* identityref
  +++-rw stats-enable? boolean
  +++-rw constants
  | ... 
  +++-rw interfaces* [reference]
  | ... 
  +++-rw hmac* [algorithm]
  | ... 
  +++-rw dtls* [name]
  ... 
augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
  +++-ro routes* [prefix]
  +++-ro prefix inet:ip-prefix
  +++-ro router-id? binary
  +++-ro neighbor? leafref
  +++-ro (metric)
  | ... 
  +++-ro seqno? uint16
  +++-ro next-hop? inet:ip-address
  +++-ro feasible? boolean
  +++-ro selected? boolean

The interfaces subtree describes attributes such as interface object
that is being referenced, the type of link as enumerated by Babel
Link Types, and whether the interface is enabled or not.

The constants subtree describes the UDP port used for sending and
receiving Babel messages, and the multicast group used to send and
receive announcements on IPv6.

The routes subtree describes objects such as the prefix for which the
route is advertised, a reference to the neighboring route, and next-
hop address.

Jethanandani & Stark Expires September 6, 2019 [Page 4]
Finally, for security two subtree are defined. The hmac subtree which refers to parameters related to HMAC security mechanism. The boolean flag apply-all indicates whether HMAC mechanism is applicable for all interfaces or just for interfaces listed in the leaf-list ‘interfaces’. The dtls subtree refers to parameters related to DTLS security mechanism. Similar to the HMAC mechanism, the boolean flag apply-all indicates whether DTLS mechanism is applicable for all interfaces or just for interfaces listed in the leaf-list ‘interfaces’.

2.2. YANG Module

This module augments A YANG Data Model for Interface Management [RFC8343], YANG Routing Management [RFC8349], and imports definitions from Common YANG Data Types [RFC6991].

module: ietf-babel
    augment /rt:routing/rt:control-plane-protocols
        /rt:control-plane-protocol:
            +++-rw babel!
                +++-ro version?       string
                +++-rw enable?         boolean
                +++-ro router-id       binary
                +++-rw link-type*       identityref
                +++-ro sequence-number? uint16
                +++-rw metric-comp-algorithms* identityref
                +++-rw security-supported* identityref
                +++-rw hmac-enable?     boolean
                +++-rw hmac-algorithms* identityref
                +++-rw dtls-enable?     boolean
                +++-rw dtls-cert-types* identityref
                +++-rw stats-enable?    boolean
                +++-rw constants
                    +++-rw udp-port?      inet:port-number
                    +++-rw mcast-group?   inet:ip-address
                +++-rw interfaces* [reference]
                    +++-rw reference       if:interface-ref
                    +++-rw enable?         boolean
                    +++-rw link-type?      identityref
                    +++-rw metric-algorithm? identityref
                    +++-ro mcast-hello-seqno? uint16
                    +++-ro mcast-hello-interval? uint16
                    +++-rw update-interval? uint16
                    +++-rw packet-log-enable? boolean
                    +++-rw packet-log?     inet:uri
                    +++-ro stats
                        +++-ro sent-mcast-hello? yt:counter32
++-ro sent-mcast-update?  yt:counter32
++-ro received-packets?  yt:counter32
+++x reset
    +++w input
        |    +++w reset-at?  yt:date-and-time
    +++ro output
        +++ro reset-finished-at?  yt:date-and-time
+++rw neighbor-objects* [neighbor-address]
    +++rw neighbor-address  inet:ip-address
    +++rw hello-mcast-history?  string
    +++rw hello-ucast-history?  string
    +++rw txcost?  int32
    +++rw exp-mcast-hello-seqno?  uint16
    +++rw exp-ucast-hello-seqno?  uint16
    +++rw ucast-hello-seqno?  uint16
    +++rw ucast-hello-interval?  uint16
    +++rw rxcost?  int32
    +++rw cost?  int32
+++ro stats
    ++-ro sent-ucast-hello?  yt:counter32
    ++-ro sent-ucast-update?  yt:counter32
    ++-ro sent-ihu?  yt:counter32
    ++-ro received-hello?  yt:counter32
    ++-ro received-update?  yt:counter32
    ++-ro received-ihu?  yt:counter32
+++x reset
    +++w input
        |    +++w reset-at?  yt:date-and-time
    +++ro output
        +++ro reset-finished-at?  yt:date-and-time
+++rw hmac* [algorithm]
    +++rw algorithm  identityref
    +++rw verify  boolean
    +++rw apply-all  boolean
    +++rw interfaces*  if:interface-ref
    +++rw hmac-keys* [name]
        +++rw name  string
        +++rw use-sign  boolean
        +++rw use-verify  boolean
        +++rw value  binary
        +++x test
            +++w input
                |            +++w test-string  binary
            +++ro output
                +++ro resulting-hash  binary
+++rw dtls* [name]
    +++rw name  string
    +++rw apply-all  boolean
---rw interfaces*  if:interface-ref
+-rw cached-info?  boolean
+-rw cert-prefer*  identityref
+-rw certs* [name]
   +-rw name  string
   +-rw value  string
   +-rw type  identityref
   +-rw private-key  binary
   --x test
      +--w input
         |   +--w test-string  binary
      +--ro output
         +--ro resulting-hash  binary
augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
   +--ro routes* [prefix]
      +--ro prefix  inet:ip-prefix
      +--ro router-id?  binary
      +--ro neighbor?  leafref
      +--ro (metric)
         |   +--:(received-metric)
         |      +--ro received-metric?  uint16
         |   +--:(calculated-metric)
         |      +--ro calculated-metric?  uint16
      +--ro seqno?  uint16
      +--ro next-hop?  inet:ip-address
      +--ro feasible?  boolean
      +--ro selected?  boolean

<CODE BEGINS> file "ietf-babel@2019-03-07.yang"

module ietf-babel {
   yang-version 1.1;
   namespace "urn:ietf:params:xml:ns:yang:ietf-babel";
   prefix babel;

   import ietf-yang-types {
      prefix yt;
      reference
         "RFC 6991 - Common YANG Data Types.";
   }
   import ietf-inet-types {
      prefix inet;
      reference
         "RFC 6991 - Common YANG Data Types.";
   }
   import ietf-interfaces {
      prefix if;
      reference

Jethanandani & Stark Expires September 6, 2019 [Page 7]
"RFC 8343 - A YANG Data Model for Interface Management";
}
import ietf-routing {
    prefix "rt";
    reference
        "RFC 8349 - YANG Routing Management";
}
organization
    "IETF Babel routing protocol Working Group";
contact
    "WG Web: http://tools.ietf.org/wg/babel/
    WG List: babel@ietf.org
    Editor: Mahesh Jethanandani
    mjethanandani@gmail.com
    Editor: Barbara Stark
    bs7652@att.com";
description
    "This YANG module defines a model for the Babel routing
    protocol.

    Copyright (c) 2018 IETF Trust and the persons identified as
    the document authors. All rights reserved. Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD
    License set forth in Section 4.c of the IETF Trust’s Legal
    Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";
revision 2019-03-07 {
    description
        "Initial version.";
    reference
        "RFC XXX: Babel YANG Data Model.";
}
/*
 * Identities
 */
identity link-type {
    description

"Base identity from which all Babel Link Types are derived."
}

identity ethernet {
    base "link-type";
    description
        "Ethernet link type for Babel Routing Protocol.";
}

identity other {
    base "link-type";
    description
        "Other link type for Babel Routing Protocol.";
}

identity tunnel {
    base "link-type";
    description
        "Tunnel link type for Babel Routing Protocol.";
}

identity wireless {
    base "link-type";
    description
        "Wireless link type for Babel Routing Protocol.";
}

identity moca {
    base "link-type";
    description
        "Multimedia over Coax Alliance.";
}

identity g-hn-over-coax {
    base "link-type";
    description
        "G.hn over coax.";
    reference
        "G.9960: Unified high-speed wireline-base home networking transceivers.";
}

identity g-hn-over-powerline {
    base "link-type";
    description
        "G.hn over powerline.";
    reference
        "G.9960: Unified high-speed wireline-base home networking transceivers.";
}

identity home-plug {
    base "link-type";
    description
        "HomePlug Power Alliance.";
reference
  "IEEE 1901: HD-PC";
}
identity iee-802-15 {  
  base "link-type";
  description
    "Wireless Personal Area Networks (WPAN).";
  reference
    "IEEE 802.15: Wireless Personal Area Networks (WPAN).";
}

identity metric-comp-algorithms {  
  description
    "Base identity from which all Babel metric comp algorithms
     are derived.";
}
identity k-out-of-j {  
  base "metric-comp-algorithms";
  description
    "k-out-of-j algorithm.";
}
identity etx {  
  base "metric-comp-algorithms";
  description
    "Expected Transmission Count.";
}

/*
 * Babel security type identities
 */
identity security-supported {  
  description
    "Base identity from which all Babel security types are
     derived.";
}

identity hmac {  
  base security-supported;
  description
    "HMAC supported.";
}

identity dtls {  
  base security-supported;
  description
    "Datagram Transport Layer Security (DTLS) supported.";
  reference
    "RFC 6347, Datagram Transport Layer Security Version 1.2.";
identity hmac-algorithms {
    description "Base identity for all Babel HMAC algorithms.";
}

identity hmac-sha256 {
    base hmac-algorithms;
    description "HMAC-SHA256 algorithm supported.";
}

identity blake2s {
    base hmac-algorithms;
    description "BLAKE2s algorithm supported.";
    reference "RFC 7693, The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC).";
}

identity dtls-cert-types {
    description "Base identity for Babel DTLS certificate types.";
}

identity x-509 {
    base dtls-cert-types;
    description "X.509 certificate type.";
}

identity raw-public-key {
    base dtls-cert-types;
    description "Raw Public Key type.";
}

/*
 * Babel routing protocol identity.
 */
identity babel {
    base "rt:control-plane-protocol";
    description
        "Babel routing protocol";
}

/*
 * Features
 */

/*
 * Features supported
 */

/*
 * Typedefs
 */

/*
 * Groupings
 */
grouping routes {
    list routes {
        key "prefix";

        leaf prefix {
            type inet:ip-prefix;
            description
                "Prefix (expressed in ip-address/prefix-length format) for
                which this route is advertised.";
            reference
                "RFC ZZZZ, Babel Information Model, Section 3.6.";
        }

        leaf router-id {
            type binary;
            description
                "router-id of the source router for which this route is
                advertised.";
            reference
                "RFC ZZZZ, Babel Information Model, Section 3.6.";
        }

        leaf neighbor {
            type leafref {
                path "/rt:routing/rt:control-plane-protocols/" +
                    "rt:control-plane-protocol/babel/interfaces/" +
                    "neighbor-objects/neighbor-address";
            }
        }
    }
}
choice metric {
  mandatory "true";
  leaf received-metric {
    type uint16;
    description "The metric with which this route was advertised by the neighbor, or maximum value (infinity) to indicate a the route was recently retracted and is temporarily unreachable. This metric will be 0 (zero) if the route was not received from a neighbor but was generated through other means. Either babel-route-calculated-metric or babel-route-received-metric MUST be provided."
    reference "RFC ZZZZ, Babel Information Model, Section 3.6, draft-ietf-babel-rfc6126bis, The Babel Routing Protocol, Section 3.5.5."
  }
  leaf calculated-metric {
    type uint16;
    description "A calculated metric for this route. How the metric is calculated is implementation-specific. Maximum value (infinity) indicates the route was recently retracted and is temporarily unreachable. Either babel-route-calculated-metric or babel-route-received-metric MUST be provided."
    reference "RFC ZZZZ, Babel Information Model, Section 3.6, draft-ietf-babel-rfc6126bis, The Babel Routing Protocol, Section 3.5.5."
  }
}

description "Either babel-route-calculated-metric or babel-route-received-metric MUST be provided.";
reference "RFC ZZZZ, Babel Information Model, Section 3.6, draft-ietf-babel-rfc6126bis, The Babel Routing Protocol, Section 3.5.5.";
leaf seqno {
  type uint16;
  description
    "The sequence number with which this route was advertised.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

leaf next-hop {
  type inet:ip-address;
  description
    "The next-hop address of this route. This will be empty if
    this route has no next-hop address.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

leaf feasible {
  type boolean;
  description
    "A boolean flag indicating whether this route is feasible.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6,
    draft-ietf-babel-rfc6126bis, The Babel Routing Protocol,
    Section 3.5.1.";
}

leaf selected {
  type boolean;
  description
    "A boolean flag indicating whether this route is selected,
    i.e., whether it is currently being used for forwarding and
    is being advertised.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

description
  "A set of babel-route-obj objects. Includes received and
  routes routes.";
reference
  "RFC ZZZZ, Babel Information Model, Section 3.1.";

description
  "Common grouping for routing used in RIB augmentation.";
}
augment "/rt:routing/rt:control-plane-protocols/" + 
"rt:control-plane-protocol" { 
  when "derived-from-or-self(rt:type, 'babel')" { 
    description 
    "Augmentation is valid only when the instance of routing type 
    is of type 'babel'."; 
  } 
  description 
  "Augment the routing module to support features such as VRF."; 
  reference 
  "YANG Routing Management, RFC 8349, Lhotka & Lindem, March 
  2018."; 
}

container babel { 
  presence "A Babel container."; 
  leaf version { 
    type string; 
    config false; 
    description 
    "The name and version of this implementation of the Babel 
    protocol."; 
    reference 
    "RFC ZZZZ, Babel Information Model, Section 3.1."; 
  }
  leaf enable { 
    type boolean; 
    default false; 
    description 
    "When written, it configures whether the protocol should be 
    enabled. A read from the <running> or <intended> datastore 
    therefore indicates the configured administrative value of 
    whether the protocol is enabled or not. 

    A read from the <operational> datastore indicates whether 
    the protocol is actually running or not, i.e. it indicates 
    the operational state of the protocol."; 
    reference 
    "RFC ZZZZ, Babel Information Model, Section 3.1."; 
  }
  leaf router-id { 
    type binary; 
    config false; 
    mandatory "true"; 
  }
description
  "Every Babel speaker is assigned a router-id, which is an
  arbitrary string of 8 octets that is assumed to be unique
  across the routing domain";
reference
  "RFC ZZZZ, Babel Information Model, Section 3.1,
  rfc6126bis, The Babel Routing Protocol. Section 3.";
}

leaf-list link-type {
  type identityref {
    base "link-type";
  }
  description
    "Link types supported by this implementation of Babel.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf sequence-number {
  type uint16;
  config false;
  description
    "Sequence number included in route updates for routes
    originated by this node.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf-list metric-comp-algorithms {
  type identityref {
    base "metric-comp-algorithms";
  }
  description
    "List of cost compute algorithms supported by this
    implementation of Babel.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf-list security-supported {
  type identityref {
    base "security-supported";
  }
  description
    "Babel security mechanism used by this implementation or
    per interface.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}
leaf hmac-enable {
  type boolean;
  description
    "Indicates whether the HMAC security mechanism is enabled (true) or disabled (false).";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf-list hmac-algorithms {
  type identityref {
    base hmac-algorithms;
  }
  description
    "List of supported HMAC computation algorithms. Possible values include 'HMAC-SHA256', 'BLAKE2s'.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf dtls-enable {
  type boolean;
  description
    "Indicates whether the DTLS security mechanism is enabled (true) or disabled (false).";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf-list dtls-cert-types {
  type identityref {
    base dtls-cert-types;
  }
  description
    "List of supported DTLS certificate types. Possible values include 'X.509' and 'RawPublicKey'.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

leaf stats-enable {
  type boolean;
  description
    "Indicates whether statistics collection is enabled (true) or disabled (false) on all interfaces, including
neighbor-specific statistics (babel-nbr-stats)."

}  

container constants {
  leaf udp-port {
    type inet:port-number;
    default "6696";
    description
      "UDP port for sending and receiving Babel messages. The
default port is 6696."
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.2.";
  }

  leaf mcast-group {
    type inet:ip-address;
    default "ff02:0:0:0:0:0:1:6";
    description
      "Multicast group for sending and receiving multicast
announcements on IPv6."
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.2.";
  }

  description
    "Babel Constants object.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.1.";
}

list interfaces {
  key "reference";

  leaf reference {
    type if:interface-ref;
    description
      "Reference to an interface object as defined by the data
model (e.g., YANG, BBF TR-181); data model is assumed to
allow for referencing of interface objects which may be at
any layer (physical, Ethernet MAC, IP, tunneled IP, etc.).
Referencing syntax will be specific to the data model. If
there is no set of interface objects available, this should
be a string that indicates the interface name used by the
underlying operating system.";
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.3.";
  }

  leaf enable {

  }
leaf link-type {
  type identityref {
    base link-type;
  }
  default "ethernet";
  description
    "Indicates the type of link. Set of values of supported
    link types where the following enumeration values MUST
    be supported when applicable: 'ethernet', 'wireless',
    'tunnel', and 'other'. Additional values MAY be
    supported.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

leaf metric-algorithm {
  type identityref {
    base metric-comp-algorithms;
  }
  default "k-out-of-j";
  description
    "Indicates the metric computation algorithm used on this
    interface. The value MUST be one of those listed in the
    babel-information-obj babel-metric-comp-algorithms
    parameter.";
}

leaf mcast-hello-seqno {
  type uint16;
  config false;
  description
    "The current sequence number in use for multicast hellos
    sent on this interface.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

leaf mcast-hello-interval {

leaf update-interval {
    type uint16;
    units centiseconds;
    description
        "The current update interval in use for this interface. Units are centiseconds.";
    reference
        "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

leaf packet-log-enable {
    type boolean;
    description
        "If true, logging of babel packets received on this interface is enabled; if false, babel packets are not logged.";
    reference
        "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

leaf packet-log {
    type inet:uri;
    description
        "A reference or url link to a file that contains a timestamped log of packets received and sent on babel-udp-port on this interface. The [libpcap] file format with .pcap file extension SHOULD be supported for packet log files. Logging is enabled / disabled by packet-log-enable.";
    reference
        "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

container stats {
    config false;
    leaf sent-mcast-hello {
        type yt:counter32;
        description
            "A count of the number of multicast Hello packets sent
leaf sent-mcast-update {
  type yt:counter32;
  description "A count of the number of multicast update packets sent on this interface.";
  reference "RFC ZZZZ, Babel Information Model, Section 3.4.";
}

leaf received-packets {
  type yt:counter32;
  description "A count of the number of Babel packets received on this interface.";
  reference "RFC ZZZZ, Babel Information Model, Section 3.4.";
}

action reset {
  input {
    leaf reset-at {
      type yt:date-and-time;
      description "The time when the reset was issued.";
    }
  }
  output {
    leaf reset-finished-at {
      type yt:date-and-time;
      description "The time when the reset finished.";
    }
  }
}

description "Statistics collection object for this interface.";
reference "RFC ZZZZ, Babel Information Model, Section 3.3.";

list neighbor-objects {
  key "neighbor-address";

  leaf neighbor-address {

type inet:ip-address;
description
"IPv4 or v6 address the neighbor sends packets from."
reference
"RFC ZZZZ, Babel Information Model, Section 3.5."
}

leaf hello-mcast-history {
  type string;
description
"The multicast Hello history of whether or not the multicast Hello packets prior to babel-exp-mcast-hello-seqno were received, with a '1' for the most recent Hello placed in the most significant bit and prior Hellos shifted right (with '0' bits placed between prior Hellos and most recent Hello for any not-received Hellos); represented as a string using utf-8 encoded hex digits where a '1' bit = Hello received and a '0' bit = Hello not received."
reference
"RFC ZZZZ, Babel Information Model, Section 3.5."
}

leaf hello-ucast-history {
  type string;
description
"The unicast Hello history of whether or not the unicast Hello packets prior to babel-exp-ucast-hello-seqno were received, with a '1' for the most recent Hello placed in the most significant bit and prior Hellos shifted right (with '0' bits placed between prior Hellos and most recent Hello for any not-received Hellos); represented as a string using utf-8 encoded hex digits where a '1' bit = Hello received and a '0' bit = Hello not received."
reference
"RFC ZZZZ, Babel Information Model, Section 3.5."
}

leaf txcost {
  type int32;
default "0";
description
"Transmission cost value from the last IHU packet received from this neighbor, or maximum value (infinity) to indicates the IHU hold timer for this neighbor has expired description."
reference
leaf exp-mcast-hello-seqno {
  type uint16;
  default "0";
  description
    "Expected multicast Hello sequence number of next Hello to be received from this neighbor; if multicast Hello packets are not expected, or processing of multicast packets is not enabled, this MUST be 0.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5.";
}

leaf exp-ucast-hello-seqno {
  type uint16;
  default "0";
  description
    "Expected unicast Hello sequence number of next Hello to be received from this neighbor; if unicast Hello packets are not expected, or processing of unicast packets is not enabled, this MUST be 0.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5.";
}

leaf ucast-hello-seqno {
  type uint16;
  description
    "Expected unicast Hello sequence number of next Hello to be received from this neighbor. If unicast Hello packets are not expected, or processing of unicast packets is not enabled, this MUST be 0.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5.";
}

leaf ucast-hello-interval {
  type uint16;
  units centiseconds;
  description
    "The current interval in use for unicast hellos sent to this neighbor. Units are centiseconds.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5.";
}
leaf rxcost {
  type int32;
  description
    "Reception cost calculated for this neighbor. This value
    is usually derived from the Hello history, which may be
    combined with other data, such as statistics maintained
    by the link layer. The rxcost is sent to a neighbor in
    each IHU."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5."
}

leaf cost {
  type int32;
  description
    "Link cost is computed from the values maintained in
    the neighbor table. The statistics kept in the neighbor
    table about the reception of Hellos, and the txcost
    computed from received IHU packets."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.5."
}

container stats {
  config false;
  leaf sent-ucast-hello {
    type yt:counter32;
    description
      "A count of the number of unicast Hello packets sent
      to this neighbor."
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.6."
  }

  leaf sent-ucast-update {
    type yt:counter32;
    description
      "A count of the number of unicast update packets sent
      to this neighbor."
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.6."
  }

  leaf sent-ihu {
    type yt:counter32;
    description
      "A count of the number of IHU packets sent to this
      neighbor."
  }
}
leaf received-hello {
  type yt:counter32;
  description
    "A count of the number of Hello packets received from
    this neighbor.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

leaf received-update {
  type yt:counter32;
  description
    "A count of the number of update packets received
    from this neighbor.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

leaf received-ihu {
  type yt:counter32;
  description
    "A count of the number of IHU packets received from
    this neighbor.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.6.";
}

action reset {
  input {
    leaf reset-at {
      type yt:date-and-time;
      description
        "The time the reset was issued.";
    }
  }
  output {
    leaf reset.finished-at {
      type yt:date-and-time;
      description
        "The time when the reset operation finished.";
    }
  }
}

description
"Statistics collection object for this neighbor.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.6.";
}
description
"A set of Babel Neighbor Object.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.5.";
}
description
"A set of Babel Interface objects.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.3.";
}
list hmac {
key "algorithm";
leaf algorithm {

type identityref {
base hmac-algorithms;
}
description
"The name of the HMAC algorithm this object instance uses. The value MUST be the same as one of the enumerations listed in the babel-hmac-algorithms parameter.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.8.";
}
leaf verify {

type boolean;
mandatory "true";
description
"A Boolean flag indicating whether HMAC hashes in incoming Babel packets are required to be present and are verified. If this parameter is 'true', incoming packets are required to have a valid HMAC hash.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.8.";
}
leaf apply-all {

type boolean;
mandatory "true";
description
"A Boolean flag indicating whether this babel-hmac instance is to be used for all interfaces. If 'true',

this instance applies to all interfaces and the babel-hmac-interfaces parameter is ignored. If babel-hmac-apply-all is ‘true’, there MUST NOT be other instances of the babel-hmac object. If ’false’, the babel-hmac-interfaces parameter determines which interfaces this instance applies to.”;
reference
"RFC ZZZZ, Babel Information Model, Section 3.8.";
}
leaf-list interfaces {
type if:interface-ref;
min-elements "1";
description
"List of references to the babel-interfaces entries this babel-hmac entry applies to. This parameter is ignored if babel-hmac-apply-all is ‘true’. An interface MUST NOT be listed in multiple instances of the babel-hmac object.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.8.";
}
list hmac-keys {
key "name";
min-elements "1";
leaf name {
type string;
mandatory "true";
description
"A unique name for this HMAC key that can be used to identify the key in this object instance, since the key value is not allowed to be read. This value can only be provided when this instance is created, and is not subsequently writable.";
reference
"RFC ZZZZ, Babel Information Model, Section 3.9.";
}
leaf use-sign {
type boolean;
mandatory "true";
description
"Indicates whether this key value is used to sign sent Babel packets. Sent packets are signed using this key if the value is ‘true’. If the value is ‘false’, this key is not used to sign sent Babel packets.";
leaf use-verify {
  type boolean;
  mandatory "true";
  description
    "Indicates whether this key value is used to verify incoming Babel packets. This key is used to verify incoming packets if the value is 'true'. If the value is 'false', no HMAC is computed from this key for comparing an incoming packet."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.9.";
}

leaf value {
  type binary;
  mandatory "true";
  description
    "The value of the HMAC key. An implementation MUST NOT allow this parameter to be read. This can be done by always providing an empty string, or through permissions, or other means. This value can only be provided when this instance is created, and is not subsequently writable."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.9.";
}

action test {
  input {
    leaf test-string {
      type binary;
      mandatory "true";
      description
        "The test string on which this test has to be performed."
    }
  }
  output {
    leaf resulting-hash {
      type binary;
      mandatory "true";
      description
        "An operation that allows the HMAC key and hash algorithm to be tested to see if they produce an
  }
expected outcome. Input to this operation is a binary string. The implementation is expected to create a hash of this string using the babel-hmac-key-value and the babel-hmac-algorithm. The output of this operation is the resulting hash, as a binary string.

reference
"RFC ZZZZ, Babel Information Model, Section 3.9.";
}
}
description
"A set of babel-hmac-keys-obj objects."
reference
"RFC ZZZZ, Babel Information Model, Section 3.8.";
}
description
"A babel-hmac-obj object. If this object is implemented, it provides access to parameters related to the HMAC security mechanism."
reference
"RFC ZZZZ, Babel Information Model, Section 3.1.";
}
list dtls {
  key "name";

  leaf name {
    type string;
    description
    "TODO: This attribute does not exist in the model, but is needed for this model to work.";
  }

  leaf apply-all {
    type boolean;
    mandatory "true";
    description
    "A Boolean flag indicating whether this babel-dtls instance is to be used for all interfaces. If 'true', this instance applies to all interfaces and the babel-dtls-interfaces parameter is ignored. If babel-dtls-apply-all is 'true', there MUST NOT be other instances of the babel-dtls object. If 'false', the babel-dtls-interfaces parameter determines which interfaces this instance applies to.";
    reference
    "RFC ZZZZ, Babel Information Model, Section 3.10.";
  }
}
leaf-list interfaces {
  type if:interface-ref;
  min-elements "1";
  description
    "List of references to the babel-interfaces entries this
    babel-dtls entry applies to. This parameter is ignored
    if babel-dtls-apply-all is 'true'. An interface MUST NOT
    be listed in multiple instances of the babel-dtls object.
    If this list is empty, then it applies to all
    interfaces."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.10.";
}

leaf cached-info {
  type boolean;
  description
    "Indicates whether the cached_info extension is included
    in ClientHello and ServerHello packets. The extension
    is included if the value is 'true'."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.10.";
}

leaf-list cert-prefer {
  type identityref {
    base dtls-cert-types;
  }
  ordered-by user;
  description
    "List of supported certificate types, in order of
    preference. The values MUST be among those listed in
    the babel-dtls-cert-types parameter. This list is used
    to populate the server_certificate_type extension in
    a Client Hello. Values that are present in at least one
    instance in the babel-dtls-certs object with a non-empty
    babel-cert-private-key will be used to populate the
    client_certificate_type extension in a Client Hello."
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.10.";
}

list certs {
  key "name";
  min-elements "1";

leaf name {
  type string;
  description
    "A unique name that identifies the cert in the list.";
}

leaf value {
  type string;
  mandatory "true";
  description
    "The DTLS certificate in PEM format [RFC7468]. This value can only be provided when this instance is created, and is not subsequently writable.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.11.";
}

leaf type {
  type identityref {
    base dtls-cert-types;
  }
  mandatory "true";
  description
    "The name of the certificate type of this object instance. The value MUST be the same as one of the enumerations listed in the babel-dtls-cert-types parameter. This value can only be provided when this instance is created, and is not subsequently writable.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.11.";
}

leaf private-key {
  type binary;
  mandatory "true";
  description
    "The value of the private key. If this is non-empty, this certificate can be used by this implementation to provide a certificate during DTLS handshaking. An implementation MUST NOT allow this parameter to be read. This can be done by always providing an empty string, or through permissions, or other means. This value can only be provided when this instance is created, and is not subsequently writable.";
  reference
    "RFC ZZZZ, Babel Information Model, Section 3.11.";
}
action test {
  input {
    leaf test-string {
      type binary;
      mandatory "true";
      description
        "The test string on which this test has to be
        performed.";
    }
  }
  output {
    leaf resulting-hash {
      type binary;
      mandatory "true";
      description
        "The output of this operation is a binary string,
        and is the resulting hash computed using the
        certificate public key, and the SHA-256
        hash algorithm.";
    }
  }
}

description
  "A set of babel-dtls-keys-obj objects. This contains
  both certificates for this implementation to present
  for authentication, and to accept from others.
  Certificates with a non-empty babel-cert-private-key
  can be presented by this implementation for
  authentication.";
reference
  "RFC ZZZZ, Babel Information Model, Section 3.10.";
}

description
  "A babel-dtls-obj object. If this object is implemented,
  it provides access to parameters related to the DTLS
  security mechanism.";
reference
  "RFC ZZZZ, Babel Information Model, Section 3.1";
}

description
  "Babel Information Objects.";
reference
  "RFC ZZZZ, Babel Information Model, Section 3.";
}

augment "/rt:routing/rt:ribs/rt:rib/rt:routes/rt:route" {
  when "derived-from(rt:source-protocol, 'babel')" {
    description

"Augmentation is valid for a routes whose source protocol is Babel."
}
derivation
"Babel specific route attributes.";
uses routes;
}

<CODE ENDS>

3. IANA Considerations

This document registers one URIs and one YANG module.

3.1. URI Registrations


3.2. YANG Module Name Registration

This document registers one YANG module in the YANG Module Names registry YANG [RFC6020].

Name: ietf-babel
prefix: babel
reference: RFC XXXX

4. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocol such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF Access Control Model (NACM [RFC8341]) provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/created/deleted (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.
These are the subtrees and data nodes and their sensitivity/vulnerability:

5. Acknowledgements

6. References

6.1. Normative References

[I-D.ietf-babel-rfc6126bis]


6.2. Informative References

[I-D.ietf-babel-information-model]
Appendix A.  An Appendix

Authors' Addresses

Mahesh Jethanandani
VMware
California
USA

Email: mjethanandani@gmail.com
Abstract

This document defines an extension to the Babel routing protocol that uses symmetric delay in metric computation and therefore makes it possible to prefer lower latency links to higher latency ones.

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 https://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 12, 2019.

Copyright Notice

Copyright (c) 2019 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 (https://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 the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

The Babel routing protocol [BABEL] does not mandate a specific algorithm for computing metrics; existing implementations use a packet-loss based metric on wireless links and a simple hop-count metric on all other types of links. While this strategy works reasonably well in many networks, it fails to select reasonable routes in some topologies involving tunnels or VPNs.

Consider for example the following topology, with three routers A, B and D located in Paris and a fourth router located in Tokyo, connected through tunnels in a diamond topology.
When routing traffic from A to D, it is obviously preferable to use the local route through B, as this is likely to provide better service quality and lower monetary cost than the distant route through C. However, the existing implementations of Babel consider both routes as having the same metric, and will therefore route the traffic through C in roughly half the cases.

In this document, we specify an extension to the Babel routing protocol that enables precise measurement of the round-trip time (RTT) of a link, and allows its usage in metric computation. Since this causes a negative feedback loop, special care is needed to ensure that the resulting network is reasonably stable (Section 2.3).

We believe that this protocol may be useful in other situations than the one described above, such as when running Babel in a congested wireless mesh network or over a complex link layer that performs its own routing; the high granularity of the timestamps used (1ms) should make it easier to experiment with RTT-based metrics on this kind of link layers.

2. Protocol operation

The protocol estimates the RTT to each neighbour (Section 2.1) which it then uses for metric computation (Section 2.2).

2.1. Delay estimation

The RTT to a neighbour is estimated using an algorithm due to Mills [MILLS], originally developed for the HELLO routing protocol and later used in NTP [NTP].

A Babel speaker periodically sends a multicast Hello message over all of its interfaces (Section 3.4.1 of [BABEL]). This Hello is usually accompanied with a set of IHU messages, one per neighbour (Section 3.4.2 of [BABEL]).

In order to enable the computation of RTTs, a node A SHOULD include in every Hello that it sends a timestamp t1 (according to A’s clock). When a node B receives A’s Hello, it records in its neighbour table the timestamp t1 as well as the time t1’ according to its own (B’s) clock at which it received the packet.

When B later sends an IHU to A, it SHOULD attach to the IHU the timestamps t1 and t1’ which it has stored in its neighbour table. Additionally, it SHOULD ensure that the packet within which the IHU is sent contains a Hello TLV with an associated timestamp t2’ (according to B’s clock). Symmetrically, A will record in its neighbour table the timestamp t2’ as well as the time t2 (according
to A’s clock) at which it has received the Hello. This is illustrated in the following sequence diagram:

```
A          B
+ t1 +   /  Hello(t1)
      |       /  + t1'
      |      /  + t2'
      |     /   /  Hello(t2')
      |    /   /   IHU(t1, t1')
      |   /   /   t2'
      |  /   /   v  v
      + /   /   t2 +
      v   v
```

A then estimates the RTT between A and B as \((t2 - t1) - (t2' - t1')\).

This algorithm has a number of desirable properties. First, since there is no requirement that \(t1'\) and \(t2'\) be equal, the protocol remains asynchronous -- the only change to Babel’s message scheduling is the requirement that a packet containing an IHU also contains a Hello. Second, since only differences of timestamps according to a single clock are computed, it does not require synchronised clocks. Third, it requires very little additional state -- a node only needs to store the two timestamps associated with the last hello received from each neighbour. Finally, since it only requires piggybacking one or two timestamps on each Hello and IHU packet, it makes efficient use of network resources.

In principle, this algorithm is inaccurate in the presence of clock drift (i.e. when A’s and B’s clocks are running at different frequencies). However, \(t2' - t1'\) is usually on the order of seconds, and significant clock drift is unlikely to happen at that time scale.
2.2. Metric computation

The algorithm described in the previous section allows computing an RTT to every neighbour. How to map this value to a link cost is a local implementation matter.

Obviously, the mapping should be monotonic (larger RTTs imply larger costs). In addition, in order to enhance stability (Section 2.3), the mapping should be bounded -- above a certain RTT, all links are equally bad.

2.2.1. Example metric computation

The current implementation of Babel uses the following function for mapping RTTs to link costs, parameterised by three parameters rtt-min, rtt-max and max-rtt-penalty:

\[
\text{cost} = \begin{cases} 
C + \text{max-rtt-penalty} & \text{for } rtt < rtt\text{-min} \\
\frac{C + \text{max-rtt-penalty}}{\text{rtt-min}} & \text{for } rtt \geq rtt\text{-min} \land rtt < rtt\text{-max} \\
0 & \text{for } rtt \geq rtt\text{-max}
\end{cases}
\]

For RTTs below rtt-min, the link cost is just the nominal cost of a single hop, C. Between rtt-min and rtt-max, the cost increases linearly; above rtt-max, the constant value max-rtt-penalty is added to the nominal cost.
2.3. Stability issues

Using delay as an input to the routing metric in congested networks gives rise to a negative feedback loop: low RTT encourages traffic, which in turn causes the RTT to increase. In a congested network, such a feedback loop can cause persistent oscillations.

The current implementation of Babel uses three techniques that collaborate to limit the frequency of oscillations:

- the measured RTT is smoothed, which limits Babel’s response to short-term RTT variations;
- the mapping function is bounded, which avoids switching between congested routes;
- a hysteresis algorithm is applied to the metric before route selection, which limits the worst-case frequency of route oscillations.

These techniques are discussed in more detail in [DELAY-BASED].

2.4. Backwards and forwards compatibility

This protocol extension stores the data that it requires within sub-TLVs of Babel’s Hello and IHU TLVs. As discussed in Section 4 of [BABEL-EXT], implementations that do not understand this extension will silently ignore the sub-TLVs while parsing the rest of the TLVs that they contain. In effect, this extension supports building hybrid networks consisting of extended and unextended routers, and while such networks might suffer from sub-optimal routing, they will not suffer from blackholes or routing loops.

If a sub-TLV defined in this extension is longer than expected, the additional data is silently ignored. This provision is made in order to allow a future version of this document to extend the packet format with additional data.

3. Packet format

This extension defines the Timestamp sub-TLV [BABEL-EXT], whose Type field has value 3. This sub-TLV can be contained within a Hello sub-TLV, in which case it carries a single timestamp, or within an IHU sub-TLV, in which case it carries two timestamps.

Timestamps are encoded as 32-bit unsigned integers, expressed in units of one microsecond, counting from an arbitrary origin.
Timestamps wrap around every 4295 seconds, or slightly more than one hour.

3.1. Timestamp sub-TLV in Hello TLVs

When contained within a Hello TLV, the Timestamp sub-TLV has the following format:

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 = 3    |    Length     |      Transmit timestamp       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type       Set to 3 to indicate a Timestamp sub-TLV.
Length     The length of the body, exclusive of the Type and Length fields.
Transmit timestamp The time at which the packet containing this sub-TLV was sent, according to the sender’s clock.

If the Length field is larger than the expected 4 octets, the sub-TLV MUST be processed normally and any extra data contained in this sub-TLV MUST be silently ignored.

3.2. Timestamp sub-TLV in IHU TLVs

When contained in an IHU TLV destined for node A, the Timestamp sub-TLV has the following format:

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 = 3    |    Length     |        Origin timestamp       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |        Receive timestamp      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Fields:

Type       Set to 3 to indicate a Timestamp sub-TLV.
Length  The length of the body, exclusive of the Type and Length fields.

Origin timestamp  A copy of the transmit timestamp of the last Timestamp sub-TLV contained in a Hello TLV received from node A.

Receive timestamp  The time at which the last Hello with a Timestamp sub-TLV was received from node A according to the sender’s clock.

If the Length field is larger than the expected 8 octets, the sub-TLV MUST be processed normally and any extra data contained in this sub-TLV MUST be silently ignored.

4. IANA Considerations

IANA is instructed to add the following entry to the "Babel Sub-TLV Types" registry:

+-----+---------------+------------------+
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Timestamp</td>
<td>(this document)</td>
</tr>
</tbody>
</table>

5. Security Considerations

This extension merely adds additional timestamping data to two of the TLVs sent by a Babel router, and does not significantly change the security properties of the Babel protocol.

6. References

6.1. Normative References


6.2. Informative References

Available online from http://arxiv.org/abs/1403.3488


Authors’ Addresses

Baptiste Jonglez
ENS Lyon
France

Email: baptiste.jonglez@ens-lyon.org

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

Email: jch@irif.fr