

Network Working Group
Internet-Draft
Obsoletes: 7298 (if approved)
Updates: 6126bis (if approved)
Intended status: Standards Track
Expires: September 10, 2019

C. Do
W. Kolodziejek
J. Chroboczek
IRIF, University of Paris-Diderot
March 9, 2019

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.

Table of Contents

1. Introduction	2
1.1. Applicability	3
1.2. Assumptions and security properties	3
1.3. Specification of Requirements	4
2. Conceptual overview of the protocol	4
3. Data Structures	5
3.1. The Interface Table	6
3.2. The Neighbour table	6
4. Protocol Operation	7
4.1. HMAC computation	7
4.2. Packet Transmission	8
4.3. Packet Reception	8
4.4. Expiring per-neighbour state	11
5. Packet Format	11
5.1. HMAC TLV	11
5.2. PC TLV	12
5.3. Challenge Request TLV	12
5.4. Challenge Reply TLV	13
6. Security Considerations	13
7. IANA Considerations	14
8. Acknowledgments	15
9. References	15
9.1. Normative References	15
9.2. Informational References	16
Appendix A. Incremental deployment and key rotation	16
Appendix B. Changes from previous versions	17
B.1. Changes since draft-ietf-babel-hmac-00	17
B.2. Changes since draft-ietf-babel-hmac-01	17
B.3. Changes since draft-ietf-babel-hmac-02	17
B.4. Changes since draft-ietf-babel-hmac-03	17
Authors' Addresses	18

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:

- o 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 [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:

- o 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]);
- o 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:

- o a pair (Index, PC), where Index is a string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer;
- o a Nonce, an arbitrary string of 0 to 192 octets, and an associated challenge expiry timer.

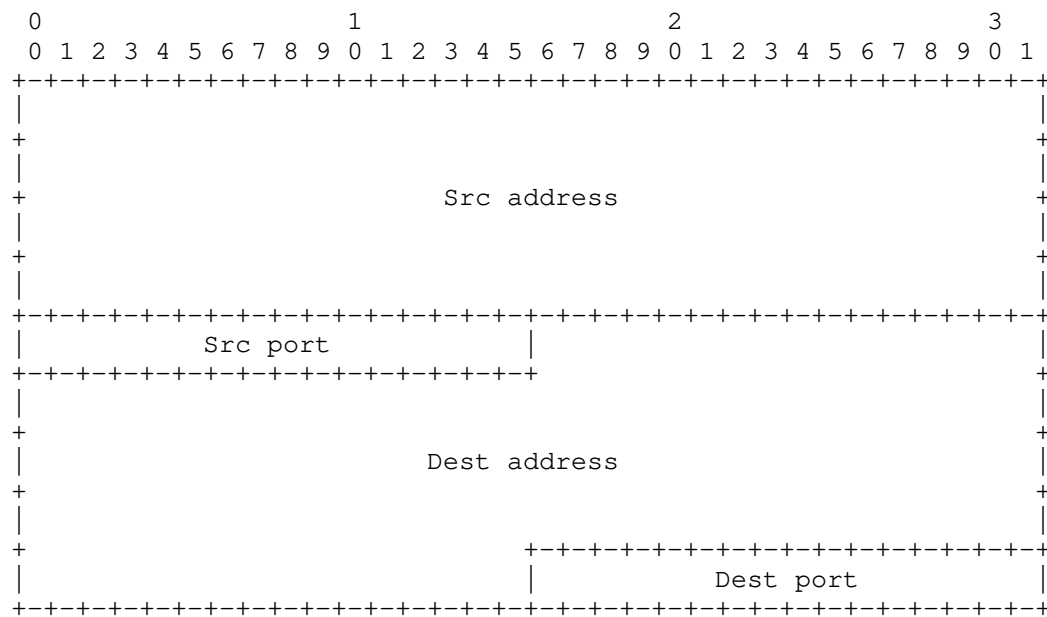
The Index and PC are initially undefined, and are managed as described in Section 4.3. The Nonce and expiry timer are initially undefined, and used as described in Section 4.3.1.1.

4. Protocol Operation

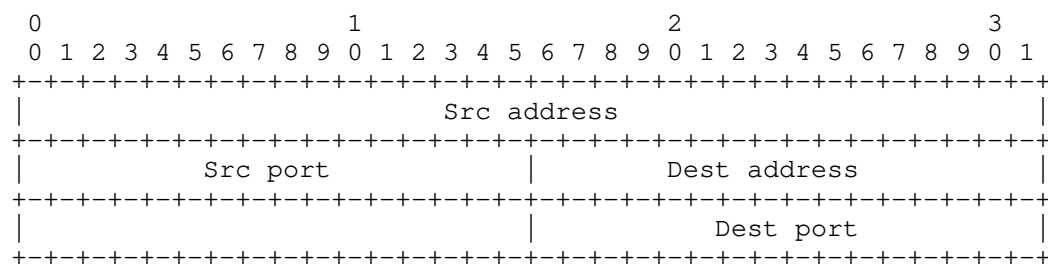
4.1. HMAC computation

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:



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



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 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:

- o 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;
- o 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:

- o 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.

- o 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.
- o 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.
- o 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.
- o 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
      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 = 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 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 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Type = 18   |   Length   |   Nonce...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```


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 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:

Type	Name	Reference
16	HMAC	this document
17	PC	this document
18	Challenge Request	this document
19	Challenge Reply	this document

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

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC6126bis] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", draft-ietf-babel-rfc6126bis-06 (work in progress), October 2018.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <<https://www.rfc-editor.org/info/rfc7693>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

9.2. Informational References

- [I-D.ietf-babel-dtls]
Decimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", draft-ietf-babel-dtls-01 (work in progress), October 2018.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<http://www.rfc-editor.org/info/rfc4086>>.
- [RFC6039] Manral, V., Bhatia, M., Jaeggli, J., and R. White, "Issues with Existing Cryptographic Protection Methods for Routing Protocols", RFC 6039, DOI 10.17487/RFC6039, October 2010, <<https://www.rfc-editor.org/info/rfc6039>>.
- [RFC7298] Ovsienko, D., "Babel Hashed Message Authentication Code (HMAC) Cryptographic Authentication", RFC 7298, DOI 10.17487/RFC7298, July 2014, <<https://www.rfc-editor.org/info/rfc7298>>.

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.

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

Network Working Group	C. Do
Internet-Draft	W. Kolodziejek
Obsoletes: 7298 (if approved)	J. Chroboczek
Intended status: Standards Track	IRIF, University of Paris-Diderot
Expires: March 8, 2021	September 4, 2020

MAC authentication for the Babel routing protocol
draft-ietf-babel-hmac-12

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 March 8, 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1. Introduction	2
1.1. Applicability	3
1.2. Assumptions and security properties	3
1.3. Specification of Requirements	4
2. Conceptual overview of the protocol	4
3. Data Structures	6
3.1. The Interface Table	6
3.2. The Neighbour table	6
4. Protocol Operation	7
4.1. MAC computation	7
4.2. Packet Transmission	8
4.3. Packet Reception	8
4.4. Expiring per-neighbour state	12
5. Incremental deployment and key rotation	12
6. Packet Format	13
6.1. MAC TLV	13
6.2. PC TLV	13
6.3. Challenge Request TLV	14
6.4. Challenge Reply TLV	14
7. Security Considerations	15
8. IANA Considerations	17
9. Acknowledgments	17
10. References	17
10.1. Normative References	17
10.2. Informational References	18
Appendix A. Changes from previous versions	19
A.1. Changes since draft-ietf-babel-hmac-00	19
A.2. Changes since draft-ietf-babel-hmac-01	19
A.3. Changes since draft-ietf-babel-hmac-02	19
A.4. Changes since draft-ietf-babel-hmac-03	19
A.5. Changes since draft-ietf-babel-hmac-04	20
A.6. Changes since draft-ietf-babel-hmac-05	20
A.7. Changes since draft-ietf-babel-hmac-06	20
A.8. Changes since draft-ietf-babel-hmac-07	20
Authors' Addresses	21

1. Introduction

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:

- o spoof a Babel packet, and redirect traffic by announcing a route with a smaller metric, a larger sequence number, 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. This document obsoletes [RFC7298].

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 Message Authentication Code (MAC) being used is invulnerable to forgery, i.e., that an attacker is unable to

generate a packet with a correct MAC without access to the secret key;

- 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 MAC 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 often enough that collisions are unlikely.

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

- o it is invulnerable to spoofing: any Babel 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 an authentic packet from C, then a node B will only accept a copy of that packet if B has accepted an older packet from C and B has received no later packet from C.

While this protocol makes 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 MAC cryptographic protection, it computes one or more MACs (one per key) which it appends to the packet. When a node A receives a packet over an interface that requires MAC cryptographic protection, it independently computes a set of MACs and compares them

to the MACs 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 MACs 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 MAC 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 MAC 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 MAC 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 of [RFC6126bis]. Implementations of this protocol **MUST** allow each interface to be provisioned with a set of one or more MAC keys and the associated MAC algorithms (see Section 4.1 for suggested algorithms, and Section 7 for suggested methods for key generation). In order to allow incremental deployment of this protocol (see Section 5), implementations **SHOULD** allow an interface to be configured in a mode in which it participates in the MAC 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 MAC authentication has been configured with two new pieces of data:

- o a set of one or more MAC keys, each associated with a given MAC algorithm;
- o 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 (see Section 7 for suggested sizes), 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:

- o a pair (Index, PC), where Index is a string of 0 to 32 octets, and PC is a 32-bit (4-octet) integer;
- o a Nonce, which is an arbitrary string of 0 to 192 octets, and an associated challenge expiry timer.

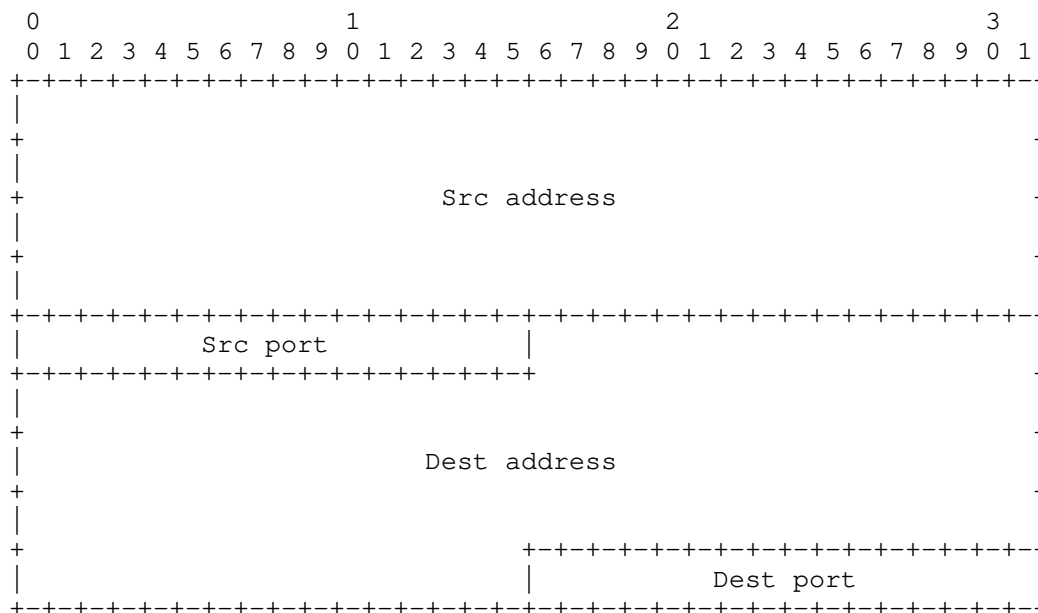
The Index and PC are initially undefined, and are managed as described in Section 4.3. The Nonce and challenge expiry timer are initially undefined, and used as described in Section 4.3.1.1.

4. Protocol Operation

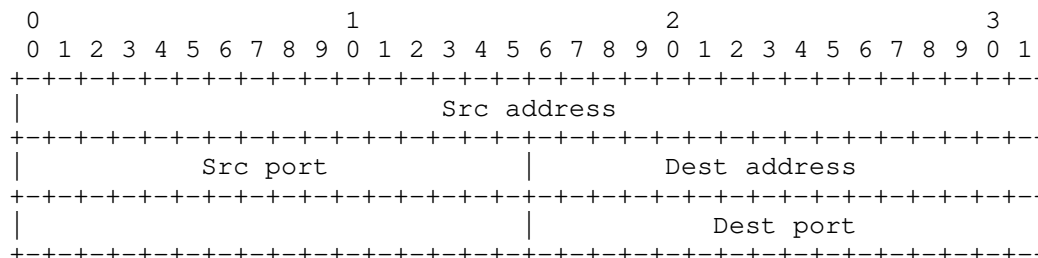
4.1. MAC computation

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

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



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



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 Babel packet including the packet header but excluding the packet trailer (from octet 0 inclusive up to (Body Length + 4) exclusive) and computes a MAC with one of the implemented 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 MAC 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 MAC protection is configured, the TLV aggregation logic MUST take into account the overhead due to PC TLVs (one in each packet) and MAC TLVs (one per configured key).

Before sending a packet, the following actions are performed:

- o 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;
- o for each key configured on the interface, a MAC is computed as specified in Section 4.1 above, and stored in a MAC 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 MAC protection, the following steps are performed before the packet is passed to normal processing:

- o First, the receiver checks whether the trailer of the received packet carries at least one MAC 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 MAC of the packet. It then compares every generated MAC against every MAC included in the packet; if there is at least one match, the packet passes the MAC 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 MAC test has passed successfully. The MAC of the packet MUST NOT be computed for each MAC TLV contained in the packet, but only once for each configured key.
- o If an entry for the sender does not exist in the Neighbour Table, it MAY be created at this point (or, alternatively, its creation can be delayed until a challenge needs to be sent, see below);
- o 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, Challenge Request and Challenge Reply TLVs. 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.
- o 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 already exists in this case), and the packet is accepted.
- o 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.

- o 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 is 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 has minimal additional cost (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 entry might need to be created at this stage), initialise the neighbour's challenge expiry timer to 30 seconds, and send 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 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.

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. This rate limiting serves two purposes. First, since a challenge may be sent in response to a packet replayed by an attacker, it limits the number of challenges that an attacker can cause a node to send. Second, it limits the number of challenges sent when there are multiple packets in flight from a single neighbour.

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 challenge sent to a multicast address MUST be silently ignored.

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.

Since a challenge reply might be caused by a replayed challenge request, a node MUST impose a rate limitation to the challenge replies it sends; the limit SHOULD default to one challenge reply for each peer every 300ms and MAY be configurable.

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 and the nonce stored in the Neighbour Table for this neighbour SHOULD be discarded; otherwise, the challenge has failed (and the nonce is not discarded).

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. Incremental deployment and key rotation

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.

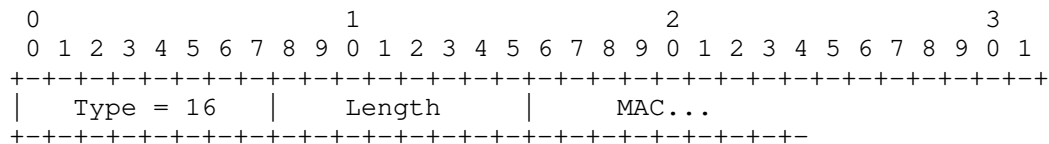
In order to perform key rotation, the new key is added to all the nodes; once this is done, both the old and the new key are sent in

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 the procedures described above, implementations of this protocol SHOULD support an interface configuration in which packets are sent authenticated but received packets are accepted without verification, and they SHOULD allow changing the set of keys associated with an interface without a restart.

6. Packet Format

6.1. MAC TLV



Fields :

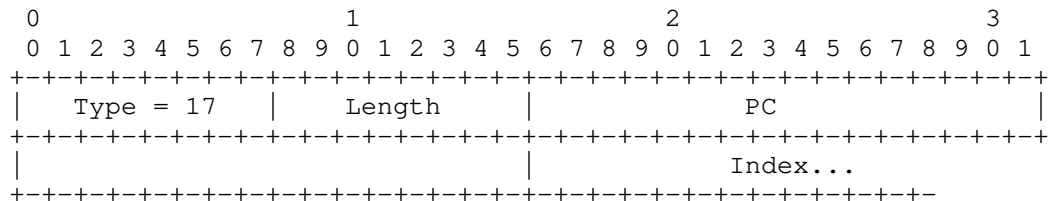
Type Set to 16 to indicate a MAC TLV.

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

MAC The body contains the MAC 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.

6.2. PC TLV



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.

6.3. 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 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  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.

6.4. 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 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.

7. 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 MAC being used is invulnerable to forgery (Section 1.1 of [RFC6039]); at the time of writing, HMAC-SHA256, 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 MAC 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, which could allow it to redirect or blackhole traffic to destinations previously advertised by the sender.

This protocol exposes large numbers of packets and their MACs to an attacker that is able to capture packets; it is therefore vulnerable to brute-force attacks. Keys must be chosen in a manner that makes them difficult to guess. Ideally, they should have a length of 32 octets (both for HMAC-SHA256 and Blake2s), and be chosen randomly. If, for some reason, it is necessary to derive keys from a human-readable passphrase, it is recommended to use a key derivation function that hampers dictionary attacks, such as PBKDF2 [RFC2898], bcrypt [BCRYPT] or scrypt [RFC7914]. In that case, only the derived keys should be communicated to the routers; the original passphrase itself should be kept on the host used to perform the key generation (e.g., an administrator's secure laptop computer).

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 MAC creates no local Babel state (Section 4.3). Reception of a replayed packet with correct MAC, on the other hand, causes a challenge to be sent; this is mitigated somewhat by requiring that challenges be rate-limited (Section 4.3.1.1).

Receiving a replayed packet with an obsolete index causes an entry to be created in the Neighbour Table, which, at first sight, makes the protocol susceptible to resource exhaustion attacks (similarly to the familiar "TCP SYN Flooding" attack [RFC4987]). However, the MAC computation includes the sender address (Section 4.1), and thus the amount of storage that an attacker can force a node to consume is limited by the number of distinct source addresses used with a single MAC key (see also Section 4 of [RFC6126bis], which mandates that the source address is a link-local IPv6 address or a local IPv4 address).

In order to make this kind of resource exhaustion attacks less effective, implementations may use a separate table of uncompleted challenges that is separate from the Neighbour Table used by the core protocol (the data structures described in Section 3.2 of [RFC6126bis] are conceptual, and any data structure that yields the same result may be used). Implementers might also consider using the fact that the nonces included in challenge requests and replies can

be fairly large (up to 192 octets), which should in principle allow encoding the per-challenge state as a secure "cookie" within the nonce itself; note however that any such scheme will need to prevent cookie replay.

8. IANA Considerations

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

Type	Name	Reference
16	MAC	this document
17	PC	this document
18	Challenge Request	this document
19	Challenge Reply	this document

9. 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 Antonin Decimo, Donald Eastlake, Toke Hoiland-Jorgensen, Florian Horn, Benjamin Kaduk, Dave Taht and Martin Vigoureux.

10. References

10.1. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC6126bis] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", draft-ietf-babel-rfc6126bis-06 (work in progress), October 2018.

- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <<https://www.rfc-editor.org/info/rfc7693>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

10.2. Informational References

- [BCRYPT] Niels, P. and D. Mazieres, "A Future-Adaptable Password Scheme", 1999.
- In Proceedings of the 1999 USENIX Annual Technical Conference.
- [I-D.ietf-babel-dtls] Decimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", draft-ietf-babel-dtls-07 (work in progress), July 2019.
- [RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <<https://www.rfc-editor.org/info/rfc2898>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<http://www.rfc-editor.org/info/rfc4086>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [RFC6039] Manral, V., Bhatia, M., Jaeggli, J., and R. White, "Issues with Existing Cryptographic Protection Methods for Routing Protocols", RFC 6039, DOI 10.17487/RFC6039, October 2010, <<https://www.rfc-editor.org/info/rfc6039>>.

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

[RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI 10.17487/RFC7914, August 2016, <<https://www.rfc-editor.org/info/rfc7914>>.

Appendix A. Changes from previous versions

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

A.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.

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

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

A.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.

A.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.

A.5. Changes since draft-ietf-babel-hmac-04

- o Use normative language in more places.

A.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.

A.7. Changes since draft-ietf-babel-hmac-06

- o Do not update RFC 6126bis, for real this time.

A.8. Changes since draft-ietf-babel-hmac-07

- o Clarify that a Neighbour Table entry may be created just after the HMAC has been computed.
- o Clarify that a Neighbour Table entry already exists when a successful Challenge Reply has been received.
- o Expand the Security Considerations section with information about resource exhaustion attacks.

A.8.1. Changes since draft-ietf-babel-hmac-08

- o Fix the size of the key to be equal to the block size, not the hash size.
- o Moved the information about incremental deployment to the body.
- o Clarified the double purpose of rate limitation.
- o Editorial changes.

A.8.2. Changes since draft-ietf-babel-hmac-09

- o Renamed HMAC to MAC throughout, relevant rewording.
- o Made it mandatory to rate-limit challenge replies in addition to requests.
- o Added discussion of key generation.
- o Added discussion of the consequences of delaying packets.

A.8.3. Changes since draft-ietf-babel-hmac-10

- o Fixed minor typos.

A.8.4. Changes since draft-ietf-babel-hmac-11

- o Clarified that the state SHOULD be discarded after a successful challenge.
- o Replaced "pre-image attack" with "forgery", this is more accurate.
- o Minor editorial changes.

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 routing protocol
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

B. Stark
AT&T
October 22, 2018

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.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
1.2. Notation	3
2. Overview	4
3. The Information Model	5
3.1. Definition of babel-information-obj	5
3.2. Definition of babel-constants-obj	6
3.3. Definition of babel-interfaces-obj	7
3.4. Definition of babel-neighbors-obj	8
3.5. Definition of babel-security-obj	10
3.6. Definition of babel-routes-obj	11
4. Common Objects	12
4.1. Definition of babel-credential-obj	12
4.2. Definition of babel-log-obj	13
5. Extending the Information Model	13
6. Security Considerations	13
7. IANA Considerations	14
8. Acknowledgements	15
9. References	15
9.1. Normative References	15
9.2. Informative References	15
Appendix A. Open Issues	17
Appendix B. Change Log	19
Author's Address	21

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:

- o enable/disable Babel
- o Constant: UDP port
- o Constant: IPv6 multicast group

- o Interface: Link type
- o Interface: External cost (must be configurable if implemented, but implementation is optional)
- o Interface: enable/disable Babel on this interface
- o Interface: enable/disable message log
- o Security: enable/disable this security mechanism
- o Security: self credentials
- o Security: trusted credentials
- o Security: enable/disable security log

Note that this overview is intended simply to be informative and is not normative. If there is any discrepancy between this overview and the detailed information model definitions in subsequent sections, the error is in this overview.

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;  
    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

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, 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. 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

- o MUST be at least 1 character and no more than 20 characters long
- o MUST contain only US-ASCII [RFC0020] letters 'A' - 'Z' and 'a' - 'z', digits '0' - '9', and hyphens ('-', ASCII 0x2D or decimal 45)
- o MUST contain at least one letter ('A' - 'Z' or 'a' - 'z')
- o MUST NOT begin or end with a hyphen
- o 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 available	(this document)
tunnel	to be used for a tunneled interface over unknown physical link	(this 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]
Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", draft-ietf-babel-rfc6126bis-05 (work in progress), May 2018.
- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [I-D.ietf-babel-dtls]
Decimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", draft-ietf-babel-dtls-01 (work in progress), October 2018.
- [I-D.ietf-babel-hmac]
Do, C., Kolodziejak, W., and J. Chroboczek, "Babel Cryptographic Authentication", draft-ietf-babel-hmac-00 (work in progress), August 2018.

- [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]
"IEEE Standard 802.3-2018 - IEEE Approved Draft Standard for Ethernet.".
- [ISO.10646]
International Organization for Standardization,
"Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8193] Burbidge, T., Eardley, P., Bagnulo, M., and J. Schoenwaelder, "Information Model for Large-Scale Measurement Platforms (LMAPs)", RFC 8193, DOI 10.17487/RFC8193, August 2017, <<https://www.rfc-editor.org/info/rfc8193>>.
- [TR-181] Broadband Forum, "Device Data Model", <<http://cwmp-data-models.broadband-forum.org/>>.

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:

v00 2017-07-03: Addressed points noted with "oops" in <https://www.ietf.org/proceedings/98/slides/slides-98-babel-babel-information-model-00.pdf>

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 routing protocol
Internet-Draft
Intended status: Informational
Expires: 12 September 2021

B.H. Stark
AT&T
M.J. Jethanandani
VMware
11 March 2021

Babel Information Model
draft-ietf-babel-information-model-14

Abstract

This Babel Information Model provides structured data elements for a Babel implementation reporting its current state and may allow limited configuration of some such data elements. This information model can be used as a basis for creating data models under various data modeling regimes. This information model only includes parameters and parameter values useful for managing Babel over IPv6.

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 12 September 2021.

Copyright Notice

Copyright (c) 2021 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.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
1.2. Notation	3
2. Overview	4
3. The Information Model	7
3.1. Definition of babel-information-obj	7
3.2. Definition of babel-constants-obj	9
3.3. Definition of babel-interface-obj	9
3.4. Definition of babel-if-stats-obj	12
3.5. Definition of babel-neighbor-obj	13
3.6. Definition of babel-route-obj	14
3.7. Definition of babel-mac-key-set-obj	16
3.8. Definition of babel-mac-key-obj	16
3.9. Definition of babel-dtls-cert-set-obj	18
3.10. Definition of babel-dtls-cert-obj	18
4. Extending the Information Model	19
5. Security Considerations	19
6. IANA Considerations	20
7. Acknowledgements	20
8. References	21
8.1. Normative References	21
8.2. Informative References	22
Authors' Addresses	22

1. Introduction

Babel is a loop-avoiding distance-vector routing protocol defined in [RFC8966]. [RFC8967] defines a security mechanism that allows Babel packets to be cryptographically authenticated, and [RFC8968] defines a security mechanism that allows Babel packets to be both authenticated and encrypted. This document describes an information model for Babel (including implementations using one or both 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 for an implementation claiming compliance with this information model. 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.

This information model only includes parameters and parameter values useful for managing Babel over IPv6. This model has no parameters or values specific to operating Babel over IPv4, even though [RFC8966] does define a multicast group for sending and listening to multicast announcements on IPv4. There is less likelihood of breakage due to inconsistent configuration and increased implementation simplicity if Babel is operated always and only over IPv6. Running Babel over IPv6 requires IPv6 at the link layer and does not need advertised prefixes, router advertisements or DHCPv6 to be present in the network. Link-local IPv6 is widely supported among devices where Babel is expected to be used. Note that Babel over IPv6 can be used for configuration of both IPv4 and IPv6 routes.

1.1. Requirements Language

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 BCP014 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 (true or false) value.
datetime	A type representing a date and time using the Gregorian calendar. The datetime format MUST conform to RFC 3339 [RFC3339] Section 5.6.
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
  +-- self-seqno
  +-- babel-metric-comp-algorithms
  +-- babel-security-supported
  +-- babel-mac-algorithms
  +-- babel-dtls-cert-types
  +-- babel-stats-enable
  +-- babel-stats-reset
  +-- babel-constants
    | +-- babel-udp-port
    | +-- babel-mcast-group
```



```
+--- babel-interfaces
|   +--- babel-interface-reference
|   +--- babel-interface-enable
|   +--- babel-interface-metric-algorithm
|   +--- babel-interface-split-horizon
|   +--- babel-mcast-hello-seqno
|   +--- babel-mcast-hello-interval
|   +--- babel-update-interval
|   +--- babel-mac-enable
|   +--- babel-if-mac-key-sets
|   +--- babel-mac-verify
|   +--- babel-dtls-enable
|   +--- babel-if-dtls-cert-sets
|   +--- babel-dtls-cached-info
|   +--- babel-dtls-cert-prefer
|   +--- babel-packet-log-enable
|   +--- babel-packet-log
|   +--- babel-if-stats
|       +--- babel-sent-mcast-hello
|       +--- babel-sent-mcast-update
|       +--- babel-sent-ucast-hello
|       +--- babel-sent-ucast-update
|       +--- babel-sent-IHU
|       +--- babel-received-packets
|   +--- babel-neighbors
|       +--- 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-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-mac-key-sets
|   +--- babel-mac-default-apply
```



```
|  +-- babel-mac-keys
|  |  +-- babel-mac-key-name
|  |  +-- babel-mac-key-use-send
|  |  +-- babel-mac-key-use-verify
|  |  +-- babel-mac-key-value
|  |  +-- babel-mac-key-algorithm
|  |  +-- babel-mac-key-test
+-- babel-dtls-cert-sets
  +-- babel-dtls-default-apply
  +-- babel-dtls-certs
    +-- babel-cert-name
    +-- babel-cert-value
    +-- babel-cert-type
    +-- babel-cert-private-key
```

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

- * enable/disable Babel
- * create/delete Babel MAC Key sets
- * create/delete Babel Certificate sets
- * enable/disable statistics collection
- * Constant: UDP port
- * Constant: IPv6 multicast group
- * Interface: enable/disable Babel on this interface
- * Interface: Metric algorithm
- * Interface: Split horizon
- * Interface: sets of MAC keys
- * Interface: verify received MAC packets
- * Interface: set of certificates for use with DTLS
- * Interface: use cached info extensions
- * Interface: preferred order of certificate types
- * Interface: enable/disable packet log

- * MAC-keys: create/delete entries
- * MAC-keys: key used for sent packets
- * MAC-keys: key used to verify packets
- * DTLS-certs: create/delete entries

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

- * MAC key values
- * DTLS private keys

Note that this overview is intended simply to be informative and is not normative. If there is any discrepancy between this overview and the detailed information model definitions in subsequent sections, the error is in this overview.

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..*>;
    [string               ro babel-mac-algorithms<1..*>;]
    [string               ro babel-dtls-cert-types<1..*>;]
    [boolean              rw babel-stats-enable;]
    [operation            babel-stats-reset;]
    babel-constants-obj   ro babel-constants;
    babel-interface-obj   ro babel-interfaces<0..*>;
    babel-route-obj       ro babel-routes<0..*>;
    [babel-mac-key-set-obj rw babel-mac-key-sets<0..*>;]
    [babel-dtls-cert-set-obj rw babel-dtls-cert-sets<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. [RFC8966] describes this as an arbitrary string of 8 octets.

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 "2-out-of-3", and "ETX". "2-out-of-3" is described in [RFC8966], section A.2.1. "ETX" is described in [RFC8966], section A.2.2.

babel-security-supported: List of supported security mechanisms. Possible values include "MAC" to indicate support of [RFC8967] and "DTLS" to indicate support of [RFC8968].

babel-mac-algorithms: List of supported MAC computation algorithms. Possible values include "HMAC-SHA256", "BLAKE2s-128" to indicate support for algorithms indicated in [RFC8967].

babel-dtls-cert-types: List of supported certificate types. Possible values include "X.509" and "RawPublicKey" to indicate support for types indicated in [RFC8968].

babel-stats-enable: Indicates whether statistics collection is enabled (true) or disabled (false) on all interfaces. When enabled, existing statistics values are not cleared and will be incremented as new packets are counted.

babel-stats-reset: An operation that resets all babel-if-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-mac-key-sets: A set of babel-mac-key-set-obj objects. If this object is implemented, it provides access to parameters related to the MAC security mechanism. An implementation MAY choose to expose this object as read-only ("ro").

babel-dtls-cert-sets: A set of babel-dtls-cert-set-obj objects. 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::1:6. An implementation MAY choose to expose this parameter as read-only ("ro").

3.3. Definition of babel-interface-obj


```
object {
  reference          ro babel-interface-reference;
  [boolean           rw babel-interface-enable;]
  string            rw babel-interface-metric-algorithm;
  [boolean           rw babel-interface-split-horizon;]
  [uint             ro babel-mcast-hello-seqno;]
  [uint             ro babel-mcast-hello-interval;]
  [uint             ro babel-update-interval;]
  [boolean           rw babel-mac-enable;]
  [reference         rw babel-if-mac-key-sets<0..*>;]
  [boolean           rw babel-mac-verify;]
  [boolean           rw babel-dtls-enable;]
  [reference         rw babel-if-dtls-cert-sets<0..*>;]
  [boolean           rw babel-dtls-cached-info;]
  [string            rw babel-dtls-cert-prefer<0..*>;]
  [boolean           rw babel-packet-log-enable;]
  [reference         ro babel-packet-log;]
  [babel-if-stats-obj ro babel-if-stats;]
  babel-neighbor-obj ro babel-neighbors<0..*>;
} babel-interface-obj;
```

babel-interface-reference: Reference to an interface object that can be used to send and receive IPv6 packets, as defined by the data model (e.g., YANG [RFC7950], BBF [TR-181]). 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-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. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-interface-split-horizon: Indicates whether or not the split

horizon optimization is used when calculating metrics on this interface. A value of true indicates split horizon optimization is used. Split horizon optimization is described in [RFC8966], section 3.7.4. An implementation MAY choose to expose this parameter as read-only ("ro").

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

babel-if-mac-keys-sets: List of references to the babel-mac entries that apply to this interface. When an interface instance is created, all babel-mac-key-sets instances with babel-mac-default-apply "true" will be included in this list. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-mac-verify A Boolean flag indicating whether MACs 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 MAC. An implementation MAY choose to expose this parameter as read-only ("ro").

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-if-dtls-cert-sets: List of references to the babel-dtls-cert-sets entries that apply to this interface. When an interface instance is created, all babel-dtls-cert-sets instances with babel-dtls-default-apply "true" will be included in this list. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-dtls-cached-info: Indicates whether the cached_info extension

(see [RFC8968] Appendix A) 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 (see [RFC8968] Appendix A) in a Client Hello. Values that are present in at least one instance in the babel-dtls-certs object of a referenced babel-dtls instance and that have a non-empty babel-cert-private-key will be used to populate the client_certificate_type extension in a Client Hello.

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. Implementations will need to carefully manage and limit memory used by packet logs.

babel-if-stats: Statistics collection object for this interface.

babel-neighbors: A set of babel-neighbor-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-sent-ucast-hello;  
    uint    ro babel-sent-ucast-update;  
    uint    ro babel-sent-IHU;  
    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-sent-ucast-hello: A count of the number of unicast Hello

packets sent on this interface.

babel-sent-ucast-update: A count of the number of unicast update packets sent on this interface.

babel-sent-IHU: A count of the number of IHU 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-neighbor-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-neighbor-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 [RFC8966], 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 [RFC8966], 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 [RFC8966], 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 NULL. This is a 16-bit unsigned integer; if the data model uses zero (0) to represent NULL values for unsigned integers, the data model MAY use a different data type that allows differentiation between zero (0) and NULL.

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 NULL. This is a 16-bit unsigned integer; if the data model uses zero (0) to represent NULL values for unsigned integers, the data model MAY use a different data type that allows differentiation between zero (0) and NULL.

babel-ucast-hello-seqno: The current sequence number in use for unicast Hellos sent to this neighbor. If unicast Hellos are not being sent, this MUST be NULL. This is a 16-bit unsigned integer; if the data model uses zero (0) to represent NULL values for unsigned integers, the data model MAY use a different data type that allows differentiation between zero (0) and NULL.

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 [RFC8966], section 3.4.3. This is a 16-bit unsigned integer.

babel-cost: The link cost, as 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.

3.6. Definition of babel-route-obj


```
object {  
  ip-address    ro babel-route-prefix;  
  uint          ro babel-route-prefix-length;  
  binary        ro babel-route-router-id;  
  reference      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-route-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: The router-id of the router that originated this route.

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 [RFC8966]). This metric will be NULL if the route was not received from a neighbor but was generated through other means. At least one of babel-route-calculated-metric and babel-route-received-metric MUST be non-NULL. Having both be non-NULL is expected for a route that is received and subsequently advertised. This is a 16-bit unsigned integer; if the data model uses zero (0) to represent NULL values for unsigned integers, the data model MAY use a different data type that allows differentiation between zero (0) and NULL.

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 [RFC8966]). At least one of `babel-route-calculated-metric` and `babel-route-received-metric` MUST be non-NULL. Having both be non-NULL is expected for a route that is received and subsequently advertised. This is a 16-bit unsigned integer; if the data model uses zero (0) to represent NULL values for unsigned integers, the data model MAY use a different data type that allows differentiation between zero (0) and NULL.

`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 [RFC8966]).

`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).

3.7. Definition of `babel-mac-key-set-obj`

```
object {  
    boolean          rw babel-mac-default-apply;  
    babel-mac-key-obj rw babel-mac-keys<0..*>;  
} babel-mac-key-set-obj;
```

`babel-mac-default-apply`: A Boolean flag indicating whether this object instance is applied to all new `babel-interface` instances, by default. If "true", this instance is applied to new `babel-interfaces` instances at the time they are created, by including it in the `babel-if-mac-key-sets` list. If "false", this instance is not applied to new `babel-interfaces` instances when they are created. An implementation MAY choose to expose this parameter as read-only ("ro").

`babel-mac-keys`: A set of `babel-mac-key-obj` objects.

3.8. Definition of `babel-mac-key-obj`


```
object {  
    string      rw babel-mac-key-name;  
    boolean     rw babel-mac-key-use-send;  
    boolean     rw babel-mac-key-use-verify;  
    binary      -- babel-mac-key-value;  
    string      rw babel-mac-key-algorithm;  
    [operation   babel-mac-key-test;]  
} babel-mac-key-obj;
```

babel-mac-key-name: A unique name for this MAC 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 **MUST NOT** be empty and can only be provided when this instance is created (i.e., it is not subsequently writable). The value **MAY** be auto-generated if not explicitly supplied when the instance is created.

babel-mac-key-use-send: Indicates whether this key value is used to compute a MAC and include that MAC in the sent Babel packet. A MAC for sent packets is computed using this key if the value is "true". If the value is "false", this key is not used to compute a MAC to include in sent Babel packets. An implementation **MAY** choose to expose this parameter as read-only ("ro").

babel-mac-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 MAC is computed from this key for comparing with the MAC in an incoming packet. An implementation **MAY** choose to expose this parameter as read-only ("ro").

babel-mac-key-value: The value of the MAC key. An implementation **MUST NOT** allow this parameter to be read. This can be done by always providing an empty string when read, or through permissions, or other means. This value **MUST** be provided when this instance is created, and is not subsequently writable. This value is of a length suitable for the associated babel-mac-key-algorithm. If the algorithm is based on the HMAC construction [RFC2104], the length **MUST** be between 0 and an upper limit that is at least the size of the output length (where "HMAC-SHA256" output length is 32 octets as described in [RFC4868]). Longer lengths **MAY** be supported but are not necessary if the management system has the ability to generate a suitably random value (e.g., by randomly generating a value or by using a key derivation technique as recommended in [RFC8967] Security Considerations). If the algorithm is "BLAKE2s-128", the length **MUST** be between 0 and 32 bytes inclusive as specified by [RFC7693].

babel-mac-key-algorithm The name of the MAC algorithm used with this

key. The value MUST be the same as one of the enumerations listed in the babel-mac-algorithms parameter. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-mac-key-test: An operation that allows the MAC key and MAC algorithm to be tested to see if they produce an expected outcome. Input to this operation are a binary string and a calculated MAC (also in the format of a binary string) for the binary string. The implementation is expected to create a MAC over the binary string using the babel-mac-key-value and the babel-mac-key-algorithm. The output of this operation is a Boolean indication that the calculated MAC matched the input MAC (true) or the MACs did not match (false).

3.9. Definition of babel-dtls-cert-set-obj

```
object {  
    boolean          rw babel-dtls-default-apply;  
    babel-dtls-cert-obj  rw babel-dtls-certs<0..*>;  
} babel-dtls-cert-set-obj;
```

babel-dtls-default-apply: A Boolean flag indicating whether this object instance is applied to all new babel-interface instances, by default. If "true", this instance is applied to new babel-interfaces instances at the time they are created, by including it in the babel-interface-dtls-certs list. If "false", this instance is not applied to new babel-interfaces instances when they are created. An implementation MAY choose to expose this parameter as read-only ("ro").

babel-dtls-certs: A set of babel-dtls-cert-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.

3.10. Definition of babel-dtls-cert-obj

```
object {  
    string          rw babel-cert-name;  
    string          rw babel-cert-value;  
    string          rw babel-cert-type;  
    binary          -- babel-cert-private-key;  
} babel-dtls-cert-obj;
```

babel-cert-name: A unique name for this certificate that can be used

to identify the certificate in this object instance, since the value is too long to be useful for identification. This value MUST NOT be empty and can only be provided when this instance is created (i.e., it is not subsequently writable). The value MAY be auto-generated if not explicitly supplied when the instance is created.

babel-cert-value: The certificate in PEM format [RFC7468]. This value MUST 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 when read, or through permissions, or other means. This value can only be provided when this instance is created, and is not subsequently writable.

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.

Misconfiguration (whether unintentional or malicious) can prevent reachability or cause poor network performance (increased latency, jitter, etc.). Misconfiguration of security credentials can cause a denial of service condition for the Babel routing protocol. The information in this model discloses network topology, which can be used to mount subsequent attacks on traffic traversing the network.

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 may be exposed through this model. This model requires that private keys and MAC keys never be exposed. Certificates used by [RFC8968] implementations use separate parameters to model the public parts (including the public key) and the private key.

MAC keys are allowed to be as short as zero-length. This is useful for testing. Network operators are RECOMMENDED to follow current best practices for key length and generation of keys related to the MAC algorithm associated with the key. Short (and zero-length) keys are highly susceptible to brute force attacks and therefore SHOULD NOT be used. See the Security Considerations section of [RFC8967] for additional considerations related to MAC keys. The fifth paragraph of [RFC8967] Security Considerations makes some specific key value recommendations that should be noted. It says that if it is necessary to derive keys from a human-readable passphrase, "only the derived keys should be communicated to the routers" and "the original passphrase itself should be kept on the host used to perform the key generation" (which would be the management system in the case of a remote management protocol). It also recommends that keys "should have a length of 32 octets (both for HMAC-SHA256 and BLAKE2s), and be chosen randomly".

This information model uses key sets and certification sets to provide a means of grouping keys and certificates. This makes it easy to use a different set per interface, the same set for one or more interfaces, have a default set in case a new interface is instantiated and to change keys and certificates as needed.

6. IANA Considerations

This document has no IANA actions.

7. Acknowledgements

Juliusz Chroboczek, Toke Hoeiland-Joergensen, David Schinazi, Antonin Decimo, 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

- [ISO.10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.
- [libpcap] Wireshark, "Libpcap File Format", 2015, <<https://wiki.wireshark.org/Development/LibpcapFileFormat>>.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <<https://www.rfc-editor.org/info/rfc4868>>.
- [RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <<https://www.rfc-editor.org/info/rfc7468>>.
- [RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <<https://www.rfc-editor.org/info/rfc7693>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [RFC8966] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", RFC 8966, DOI 10.17487/RFC8966, January 2021, <<https://www.rfc-editor.org/info/rfc8966>>.
- [RFC8967] Dô, C., Kolodziejak, W., and J. Chroboczek, "MAC Authentication for the Babel Routing Protocol", RFC 8967, DOI 10.17487/RFC8967, January 2021, <<https://www.rfc-editor.org/info/rfc8967>>.
- [RFC8968] Décimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", RFC 8968, DOI 10.17487/RFC8968, January 2021, <<https://www.rfc-editor.org/info/rfc8968>>.

8.2. Informative References

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8193] Burbridge, T., Eardley, P., Bagnulo, M., and J. Schoenwaelder, "Information Model for Large-Scale Measurement Platforms (LMAPs)", RFC 8193, DOI 10.17487/RFC8193, August 2017, <<https://www.rfc-editor.org/info/rfc8193>>.
- [TR-181] Broadband Forum, "Device Data Model", <<http://cwmp-data-models.broadband-forum.org/>>.

Authors' Addresses

Barbara Stark
AT&T
Atlanta, GA,
United States of America

Email: barbara.stark@att.com

Maresh Jethanandani
VMware
California
United States of America

Email: mjethanandani@gmail.com

Babel Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2019

M. Jethanandani
VMware
B. Stark
AT&T
March 5, 2019

YANG Data Model for Babel
draft-ietf-babel-yang-model-01

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

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.

Table of Contents

1. Introduction	2
1.1. Note to RFC Editor	2
1.2. Definitions and Acronyms	3
1.3. Tree Diagram	3
2. Babel Module	3
2.1. Information Model	3
2.2. YANG Module	5
3. IANA Considerations	33
3.1. URI Registrations	33
3.2. YANG Module Name Registration	33
4. Security Considerations	33
5. Acknowledgements	34
6. References	34
6.1. Normative References	34
6.2. Informative References	34
Appendix A. An Appendix	35
Authors' Addresses	35

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.

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

- o Revision date in model, in the format 2019-03-07 needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

1.2. Definitions and Acronyms

- o

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.

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

```



```
    "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 ieee-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.";
  }
```



```
}

/*
 * Babel HMAC algorithms identities.
 */
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).";
}

/*
 * Babel Cert Types
 */
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";
      }
    }
  }
}
```



```
    }
    description
      "Reference to the babel-neighbors entry for the neighbor
       that advertised this route.";
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.6.";
  }

  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.";
}

/*
 * Data model
```



```
*/

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, BRF 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 {
```



```
    type boolean;
    default "true";
    description
      "If true, babel sends and receives messages on this
       interface. If false, babel messages received on this
       interface are ignored and none are sent.";
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.3.";
  }

  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 {
```



```
    type uint16;
    config false;
    description
        "The current multicast hello interval in use for hellos
        sent on this interface.";
    reference
        "RFC ZZZZ, Babel Information Model, Section 3.3.";
}

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
```



```
        on this interface.";
    reference
        "RFC ZZZZ, Babel Information Model, Section 3.4.";
}

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
```



```
    "RFC ZZZZ, Babel Information Model, Section 3.5.";
}

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.";
  }
}
```



```
        reference
          "RFC ZZZZ, Babel Information Model, Section 3.6.";
      }

      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.";
    }
}
```



```
    reference
      "RFC ZZZZ, Babel Information Model, Section 3.9.";
  }

  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 bto 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.";
    }
    description
      "Babel specific route attributes.";
    uses routes;
  }
}
```

<CODE ENDS>

3. IANA Considerations

This document registers one URIs and one YANG module.

3.1. URI Registrations

URI: urn:ietf:params:xml:ns:yang:ietf-babel

3.2. YANG Module Name Registration

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

Name: ietf-babel

Namespace: urn:ietf:params:xml:ns:yang: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]
Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", draft-ietf-babel-rfc6126bis-07 (work in progress), November 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <<https://www.rfc-editor.org/info/rfc8349>>.

6.2. Informative References

- [I-D.ietf-babel-information-model]
Stark, B. and M. Jethanandani, "Babel Information Model", draft-ietf-babel-information-model-05 (work in progress), March 2019.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. An Appendix

Authors' Addresses

Mahesh Jethanandani
VMware
California
USA

Email: mjethanandani@gmail.com

Barbara Stark
AT&T
Atlanta, GA
USA

Email: barbara.stark@att.com

Babel Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 March 2022

M. Jethanandani
Kloud Services
B. Stark
AT&T
22 September 2021

YANG Data Model for Babel
draft-ietf-babel-yang-model-13

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", "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 26 March 2022.

Copyright Notice

Copyright (c) 2021 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.

Table of Contents

1. Introduction	2
1.1. Note to RFC Editor	2
1.2. Tree Diagram Annotations	3
2. Babel Module	3
2.1. Information Model	3
2.2. Tree Diagram	3
2.3. YANG Module	5
3. IANA Considerations	32
3.1. URI Registrations	32
3.2. YANG Module Name Registration	32
4. Security Considerations	32
5. Acknowledgements	34
6. References	34
6.1. Normative References	34
6.2. Informative References	35
Appendix A. Tree Diagram and Example Configurations	36
A.1. Complete Tree Diagram	36
A.2. Statistics Gathering Enabled	38
A.3. Automatic Detection of Properties	39
A.4. Override Default Properties	41
A.5. Configuring other Properties	42
Authors' Addresses	43

1. Introduction

This document defines a data model for The Babel Routing Protocol [RFC8966]. The data model is defined using YANG 1.1 [RFC7950] and is Network Management Datastore Architecture (NDMA) [RFC8342] compatible. It is based on the Babel Information Model [RFC9046]. The data model only includes data nodes that are useful for managing Babel over IPv6.

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.

- * Revision date in model, in the format 2021-09-20 needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

1.2. Tree Diagram Annotations

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 [RFC9046].

2.1. Information Model

There are a few things that should be noted between the Babel Information Model and this data module. The information model mandates the definition of some of the attributes, e.g., 'babel-implementation-version' or the 'babel-self-router-id'. These attributes are marked as read-only objects in the information module as well as in this data module. However, there is no way in the data module to mandate that a read-only attribute be present. It is up to the implementation of this data module to make sure that the attributes that are marked read-only and are mandatory are indeed present.

2.2. Tree Diagram

The following diagram illustrates a top level hierarchy of the model. In addition to the version implemented by this device, the model contains subtrees on 'constants', 'interfaces', 'mac-key-set', 'dtls', and 'routes'.


```
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
      +--ro seqno?            uint16
      +--rw statistics-enabled? boolean
      +--rw constants
      |   ...
      +--rw interfaces* [reference]
      |   ...
      +--rw mac-key-set* [name]
      |   ...
      +--rw dtls* [name]
      |   ...
      +--ro routes* [prefix]
      |   ...
```

The 'interfaces' subtree describes attributes such as the 'interface' object that is being referenced, the type of link, e.g., wired, wireless or tunnel, as enumerated by 'metric-algorithm' and 'split-horizon' 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.

Finally, for security two subtrees are defined to contain MAC keys and DTLS certificates. The 'mac-key-set' subtree contains keys used with the MAC security mechanism. The boolean flag 'default-apply' indicates whether the set of MAC keys is automatically applied to new interfaces. The 'dtls' subtree contains certificates used with DTLS security mechanism. Similar to the MAC mechanism, the boolean flag 'default-apply' indicates whether the set of DTLS certificates is automatically applied to new interfaces.

2.3. YANG Module

This YANG module augments the YANG Routing Management [RFC8349] module to provide a common framework for all routing subsystems. By augmenting the module it provides a common building block for routes, and Routing Information Bases (RIBs). It also has a reference to an interface defined by A YANG Data Model for Interface Management [RFC8343].

A router running Babel routing protocol can sometimes determine the parameters it needs to use for an interface based on the interface name. For example, it can detect that eth0 is a wired interface, and that wlan0 is a wireless interface. This is not true for a tunnel interface, where the link parameters need to be configured explicitly.

For a wired interface, it will assume 'two-out-of-three' for 'metric-algorithm', and 'split-horizon' set to true. On the other hand, for a wireless interface it will assume 'etx' for 'metric-algorithm', and 'split-horizon' set to false. However, if the wired link is connected to a wireless radio, the values can be overridden by setting 'metric-algorithm' to 'etx', and 'split-horizon' to false. Similarly, an interface that is a metered 3G link, and used for fallback connectivity needs much higher default time constants, e.g., 'mcast-hello-interval', and 'update-interval', in order to avoid carrying control traffic as much as possible.

In addition to the modules used above, this module imports definitions from Common YANG Data Types [RFC6991], and references HMAC: Keyed-Hashing for Message Authentication [RFC2104], Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec [RFC4868], The Datagram Transport Layer Security (DTLS) Version 1.3 [I-D.ietf-tls-dtls13], The Blake2 Cryptographic Hash and Message Authentication Code (MAC) [RFC7693], Babel Information Model [RFC9046], The Babel Routing Protocol [RFC8966], YANG Data Types and Groupings for Cryptography [I-D.ietf-netconf-crypto-types], Network Configuration Access Control Model [RFC8341] and MAC Authentication for Babel [RFC8967].

```
<CODE BEGINS> file "ietf-babel@2021-09-20.yang"
module ietf-babel {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-babel";
  prefix babel;

  import ietf-yang-types {
    prefix yang;
    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
    "RFC 8343: A YANG Data Model for Interface Management";
}
import ietf-routing {
  prefix rt;
  reference
    "RFC 8349: YANG Routing Management";
}
import ietf-crypto-types {
  prefix ct;
  reference
    "I-D.ietf-netconf-crypto-types: YANG Data Types and Groupings
    for Cryptographay.";
}
import ietf-netconf-acm {
  prefix nacm;
  reference
    "RFC 8341: Network Configuration Access Control Model";
}

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.

  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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2021 IETF Trust and the persons identified as authors of the code. 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 (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2021-09-20 {
  description
    "Initial version.";
  reference
    "RFC XXXX: Babel YANG Data Model.";
}

/*
 * Features
 */

feature two-out-of-three-supported {
  description
    "This implementation supports the '2-out-of-3'
    computation algorithm.";
}

feature etx-supported {
  description
    "This implementation supports the Expected Transmission Count
    (ETX) metric computation algorithm.";
}

feature mac-supported {
  description
    "This implementation supports MAC-based security.";
  reference
    "RFC 8967: MAC authentication for Babel Routing
    Protocol.";
}
```



```
feature dtls-supported {
  description
    "This implementation supports DTLS based security.";
  reference
    "RFC 8968: Babel Routing Protocol over Datagram
    Transport Layer Security.";
}

feature hmac-sha256-supported {
  description
    "This implementation supports the HMAC-SHA256 MAC algorithm.";
  reference
    "RFC 8967: MAC authentication for Babel Routing
    Protocol.";
}

feature blake2s-supported {
  description
    "This implementation supports BLAKE2s MAC algorithms.";
  reference
    "RFC 8967: MAC authentication for Babel Routing
    Protocol.";
}

feature x-509-supported {
  description
    "This implementation supports the X.509 certificate type.";
  reference
    "RFC 8968: Babel Routing Protocol over Datagram
    Transport Layer Security.";
}

feature raw-public-key-supported {
  description
    "This implementation supports the Raw Public Key certificate
    type.";
  reference
    "RFC 8968: Babel Routing Protocol over Datagram
    Transport Layer Security.";
}

/*
 * Identities
 */

identity metric-comp-algorithms {
  description
    "Base identity from which all Babel metric computation
```



```
        algorithms MUST be derived.";
    }

    identity two-out-of-three {
        if-feature "two-out-of-three-supported";
        base metric-comp-algorithms;
        description
            "2-out-of-3 algorithm.";
        reference
            "RFC 8966: The Babel Routing Protocol, Section A.2.1.";
    }

    identity etx {
        if-feature "etx-supported";
        base metric-comp-algorithms;
        description
            "Expected Transmission Count (ETX) metric computation
            algorithm.";
        reference
            "RFC 8966: The Babel Routing Protocol, Section A.2.2.";
    }

    /*
     * Babel MAC algorithms identities.
     */

    identity mac-algorithms {
        description
            "Base identity for all Babel MAC algorithms.";
    }

    identity hmac-sha256 {
        if-feature "mac-supported";
        if-feature "hmac-sha256-supported";
        base mac-algorithms;
        description
            "HMAC-SHA256 algorithm supported.";
        reference
            "RFC 4868: Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512
            with IPsec.";
    }

    identity blake2s {
        if-feature "mac-supported";
        if-feature "blake2s-supported";
        base mac-algorithms;
        description
            "BLAKE2s algorithms supported. Specifically, BLAKE2-128 is
```



```
        supported.";
    reference
        "RFC 7693: The BLAKE2 Cryptographic Hash and Message
        Authentication Code (MAC).";
}

/*
 * Babel Cert Types
 */

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

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

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

/*
 * Babel routing protocol identity.
 */

identity babel {
    base rt:routing-protocol;
    description
        "Babel routing protocol";
}

/*
 * Groupings
 */

grouping routes {
    list routes {
        key "prefix";
```



```
config false;

leaf prefix {
  type inet:ip-prefix;
  description
    "Prefix (expressed in ip-address/prefix-length format) for
    which this route is advertised.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf router-id {
  type binary {
    length 8;
  }
  description
    "router-id of the source router for which this route is
    advertised.";
  reference
    "RFC 9046: 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";
  }
  description
    "Reference to the neighbor-objects entry for the neighbor
    that advertised this route.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf received-metric {
  type union {
    type enumeration {
      enum null {
        description
          "Route was not received from a neighbor.";
      }
    }
    type uint16;
  }
  description
    "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. This metric will be NULL if the
        route was not received from a neighbor but instead was
        injected through means external to the Babel routing
        protocol. At least one of calculated-metric or
        received-metric MUST be non-NULL.";
    reference
        "RFC 9046: Babel Information Model, Section 3.6,
        RFC 8966: The Babel Routing Protocol, Section 2.1.";
}

leaf calculated-metric {
    type union {
        type enumeration {
            enum null {
                description
                    "Route has not been calculated.";
            }
        }
        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. At least one of
        calculated-metric or received-metric MUST be non-NULL.";
    reference
        "RFC 9046: Babel Information Model, Section 3.6,
        RFC 8966: The Babel Routing Protocol, Section 2.1.";
}

leaf seqno {
    type uint16;
    description
        "The sequence number with which this route was
        advertised.";
    reference
        "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf next-hop {
    type union {
        type enumeration {
            enum null {
                description
                    "Route has no next-hop address.";
            }
        }
    }
}
```



```
    }
    type inet:ip-address;
  }
  description
    "The next-hop address of this route. This will be NULL
    if this route has no next-hop address.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf feasible {
  type boolean;
  description
    "A boolean flag indicating whether this route is
    feasible.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6,
    RFC 8966, 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 9046: Babel Information Model, Section 3.6.";
}
description
  "A set of babel-route-obj objects. Contains routes known to
  this node.";
reference
  "RFC 9046: Babel Information Model, Section 3.1.";
}
description
  "Common grouping for routing used in RIB.";
}

/*
 * Data model
 */

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 a common structure
        between routing protocols.";
    reference
        "YANG Routing Management, RFC 8349, Lhotka & Lindem, March
        2018.";

    container babel {
        presence "A Babel container.";
        description
            "Babel Information Objects.";
        reference
            "RFC 9046: Babel Information Model, Section 3.";

        leaf version {
            type string;
            config false;
            description
                "The name and version of this implementation of the Babel
                protocol.";
            reference
                "RFC 9046: Babel Information Model, Section 3.1.";
        }

        leaf enable {
            type boolean;
            mandatory true;
            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 9046: Babel Information Model, Section 3.1.";
        }

        leaf router-id {
            type binary;
            must '../enable = "true"';
            config false;
            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.

The router-id is valid only if the protocol is enabled, at which time a non-zero value is assigned.";

reference

"RFC 9046: Babel Information Model, Section 3.1,
RFC 8966: The Babel Routing Protocol,
Section 3.";

}

leaf seqno {

type uint16;

config false;

description

"Sequence number included in route updates for routes
originated by this node.";

reference

"RFC 9046: Babel Information Model, Section 3.1.";

}

leaf statistics-enabled {

type boolean;

description

"Indicates whether statistics collection is enabled (true)
or disabled (false) on all interfaces. On transition to
enabled, existing statistics values are not cleared and
will be incremented as new packets are counted.";

}

container constants {

description

"Babel Constants object.";

reference

"RFC 9046: Babel Information Model, Section 3.1.";

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 9046: Babel Information Model, Section 3.2.";

}

leaf mcast-group {

type inet:ip-address;


```
    default "ff02::1:6";
    description
      "Multicast group for sending and receiving multicast
       announcements on IPv6.";
    reference
      "RFC 9046: Babel Information Model, Section 3.2.";
  }
}

list interfaces {
  key "reference";

  description
    "A set of Babel Interface objects.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.";

  leaf reference {
    type if:interface-ref;
    description
      "References the name of the interface over which Babel
       packets are sent and received.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf enable {
    type boolean;
    default "true";
    description
      "If true, babel sends and receives messages on this
       interface. If false, babel messages received on this
       interface are ignored and none are sent.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf metric-algorithm {
    type identityref {
      base metric-comp-algorithms;
    }
    mandatory true;
    description
      "Indicates the metric computation algorithm used on this
       interface. The value MUST be one of those identities
       based on 'metric-comp-algorithms'.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }
}
```



```
}

leaf split-horizon {
  type boolean;
  description
    "Indicates whether or not the split horizon optimization
     is used when calculating metrics on this interface.
     A value of true indicates the split horizon optimization
     is used.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.";
}

leaf mcast-hello-seqno {
  type uint16;
  config false;
  description
    "The current sequence number in use for multicast hellos
     sent on this interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.";
}

leaf mcast-hello-interval {
  type uint16;
  units "centiseconds";
  description
    "The current multicast hello interval in use for hellos
     sent on this interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.";
}

leaf update-interval {
  type uint16;
  units "centiseconds";
  description
    "The current update interval in use for this interface.
     Units are centiseconds.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.";
}

leaf mac-enable {
  type boolean;
  description
    "Indicates whether the MAC security mechanism is enabled
     (true) or disabled (false).";
```



```
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf-list mac-key-sets {
    type leafref {
      path "../..//mac-key-set/name";
    }
    description
      "List of references to the MAC entries that apply
       to this interface. When an interface instance is
       created, all MAC instances with default-apply 'true'
       will be included in this list.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf mac-verify {
    type boolean;
    description
      "A Boolean flag indicating whether MACs 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 MAC.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf dtls-enable {
    type boolean;
    description
      "Indicates whether the DTLS security mechanism is enabled
       (true) or disabled (false).";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }

  leaf-list dtls-certs {
    type leafref {
      path "../..//dtls/name";
    }
    description
      "List of references to the dtls entries that apply to
       this interface. When an interface instance
       is created, all dtls instances with default-apply
       'true' will be included in this list.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";
  }
```



```
}

leaf dtls-cached-info {
  type boolean;
  description
    "Indicates whether the cached_info extension is enabled.
    The extension is enabled for inclusion in ClientHello
    and ServerHello messages if the value is 'true'.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3.
    RFC 8968: Babel Routing Protocol over
    Datagram Transport Layer Security, Appendix A.";
}

leaf-list dtls-cert-prefer {
  type leafref {
    path "../dtls/certs/type";
  }
  ordered-by user;
  description
    "List of supported certificate types, in order of
    preference. The values MUST be the 'type' attribute
    in the list 'certs' of the list 'dtls'
    (../dtls/certs/type). This list is used to populate
    the server_certificate_type extension in a ClientHello.
    Values that are present in at least one instance in the
    certs object under dtls of a referenced dtls instance
    and that have a non-empty private-key will be used to
    populate the client_certificate_type extension in a
    ClientHello.";
  reference
    "RFC 9046: Babel Information Model, Section 3.3
    RFC 8968: Babel Routing Protocol over
    Datagram Transport Layer Security, Appendix A.";
}

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 9046: Babel Information Model, Section 3.3.";
}

leaf packet-log {
  type inet:uri;
```



```
    config false;
    description
      "A reference or url link to a file that contains a
       timestamped log of packets received and sent on
       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 9046: Babel Information Model, Section 3.3.";
  }

  container statistics {
    config false;
    description
      "Statistics collection object for this interface.";
    reference
      "RFC 9046: Babel Information Model, Section 3.3.";

    leaf discontinuity-time {
      type yang:date-and-time;
      mandatory true;
      description
        "The time on the most recent occasion at which any one
         or more of counters suffered a discontinuity. If no
         such discontinuities have occurred since the last
         re-initialization of the local management subsystem,
         then this node contains the time the local management
         subsystem re-initialized itself.";
    }

    leaf sent-mcast-hello {
      type yang:counter32;
      description
        "A count of the number of multicast Hello packets sent
         on this interface.";
      reference
        "RFC 9046: Babel Information Model, Section 3.4.";
    }

    leaf sent-mcast-update {
      type yang:counter32;
      description
        "A count of the number of multicast update packets sent
         on this interface.";
      reference
        "RFC 9046: Babel Information Model, Section 3.4.";
    }
  }
```



```
leaf sent-ucast-hello {
  type yang:counter32;
  description
    "A count of the number of unicast Hello packets sent
    on this interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf sent-ucast-update {
  type yang:counter32;
  description
    "A count of the number of unicast update packets sent
    on this interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf sent-ihu {
  type yang:counter32;
  description
    "A count of the number of IHU packets sent on this
    interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.6.";
}

leaf received-packets {
  type yang:counter32;
  description
    "A count of the number of Babel packets received on
    this interface.";
  reference
    "RFC 9046: Babel Information Model, Section 3.4.";
}

action reset {
  description
    "The information model [RFC 9046] defines reset
    action as a system-wide reset of Babel statistics.
    In YANG the reset action is associated with the
    container where the action is defined. In this case
    the action is associated with the statistics container
    inside an interface. The action will therefore
    reset statistics at an interface level.

    Implementations that want to support a system-wide
    reset of Babel statistics need to call this action
```



```
        for every instance of the interface.";

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

    output {
        leaf reset-finished-at {
            type yang:date-and-time;
            description
                "The time when the reset finished.";
        }
    }
}

list neighbor-objects {
    key "neighbor-address";
    config false;
    description
        "A set of Babel Neighbor Object.";
    reference
        "RFC 9046: Babel Information Model, Section 3.5.";

    leaf neighbor-address {
        type inet:ip-address;
        description
            "IPv4 or v6 address the neighbor sends packets from.";
        reference
            "RFC 9046: 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 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 of
            utf-8 encoded hex digits. A bit that is set indicates
            that the corresponding Hello was received, and a bit
```



```
        that is cleared indicates that the corresponding Hello
        was not received.";
    reference
        "RFC 9046: 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 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 9046: 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 indicate the IHU hold timer for this
        neighbor has expired description.";
    reference
        "RFC 9046: Babel Information Model, Section 3.5.";
}

leaf exp-mcast-hello-seqno {
    type union {
        type enumeration {
            enum null {
                description
                    "Multicast Hello packets are not expected, or
                    processing of multicast packets is not
                    enabled.";
            }
        }
        type uint16;
    }
    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 NULL.";
    reference
        "RFC 9046: Babel Information Model, Section 3.5.";
}

leaf exp-ucast-hello-seqno {
    type union {
        type enumeration {
            enum null {
                description
                    "Unicast Hello packets are not expected, or
                     processing of unicast packets is not enabled.";
            }
        }
        type uint16;
    }
    default null;
    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 NULL.";
    reference
        "RFC 9046: Babel Information Model, Section 3.5.";
}

leaf ucast-hello-seqno {
    type union {
        type enumeration {
            enum null {
                description
                    "Unicast Hello packets are not being sent.";
            }
        }
        type uint16;
    }
    default null;
    description
        "The current sequence number in use for unicast Hellos
         sent to this neighbor. If unicast Hellos are not being
         sent, this MUST be NULL.";
    reference
        "RFC 9046: 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 9046: Babel Information Model, Section 3.5.";
  }

  leaf rxcost {
    type uint16;
    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 9046: 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 9046: Babel Information Model, Section 3.5.";
  }
}

list mac-key-set {
  key "name";

  description
    "A MAC key set object. If this object is implemented, it
     provides access to parameters related to the MAC security
     mechanism.";
  reference
    "RFC 9046: Babel Information Model, Section 3.7.";

  leaf name {
    type string;
    description
      "A string that uniquely identifies the MAC object.";
```



```
}

leaf default-apply {
  type boolean;
  description
    "A Boolean flag indicating whether this object
    instance is applied to all new interfaces, by default.
    If 'true', this instance is applied to new babel-
    interfaces instances at the time they are created,
    by including it in the mac-key-sets list under
    the interface. If 'false', this instance is not applied
    to new interface instances when they are created.";
  reference
    "RFC 9046: Babel Information Model, Section 3.7.";
}

list keys {
  key "name";
  min-elements 1;
  description
    "A set of keys objects.";
  reference
    "RFC 9046: Babel Information Model, Section 3.8.";

  leaf name {
    type string;
    description
      "A unique name for this MAC 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 9046: Babel Information Model, Section 3.8.";
  }

  leaf use-send {
    type boolean;
    mandatory true;
    description
      "Indicates whether this key value is used to compute a
      MAC and include that MAC in the sent Babel packet. A
      MAC for sent packets is computed using this key if the
      value is 'true'. If the value is 'false', this key is
      not used to compute a MAC to include in sent Babel
      packets.";
    reference
      "RFC 9046: Babel Information Model, Section 3.8.";
  }
}
```



```
}

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 MAC is computed from this key for
    comparing an incoming packet.";
  reference
    "RFC 9046: Babel Information Model, Section 3.8.";
}

leaf value {
  nacm:default-deny-all;
  type binary;
  mandatory true;
  description
    "The value of the MAC key.

    This value is of a length suitable for the associated
    babel-mac-key-algorithm. If the algorithm is based on
    the HMAC construction [RFC2104], the length MUST be
    between 0 and an upper limit that is at least the size
    of the output length (where 'HMAC-SHA256' output
    length is 32 octets as described in [RFC4868]). Longer
    lengths MAY be supported but are not necessary if the
    management system has the ability to generate a
    suitably random value (e.g., by randomly generating a
    value or by using a key derivation technique as
    recommended in [RFC8967] Security Considerations). If
    the algorithm is 'BLAKE2s-128', the length MUST be
    between 0 and 32 bytes inclusive as specified by
    [RFC7693].";
  reference
    "RFC 9046: Babel Information Model, Section 3.8,
    RFC 2104: HMAC: Keyed-Hashing for Message
    Authentication
    RFC 4868: Using HMAC-SHA-256, HMAC-SHA-384, and
    HMAC-SHA-512 with IPsec,
    RFC 7693: The BLAKE2 Cryptographic Hash and Message
    Authentication Code (MAC).
    RFC 8967: MAC Authentication for Babel.";
}

leaf algorithm {
```



```
type identityref {
  base mac-algorithms;
}
mandatory true;
description
  "The MAC algorithm used with this key. The
   value MUST be one of the identities
   listed with the base of 'mac-algorithms'.";
reference
  "RFC 9046: Babel Information Model, Section 3.8.";
}

action test {
  description
    "An operation that allows the MAC key and MAC
     algorithm to be tested to see if they produce an
     expected outcome. Input to this operation are a
     binary string and a calculated MAC (also in the
     format of a binary string) for the binary string.
     The implementation is expected to create a MAC over
     the binary string using the value and algorithm.
     The output of this operation is a binary indication
     that the calculated MAC matched the input MAC (true)
     or the MACs did not match (false).";
  reference
    "RFC 9046: Babel Information Model, Section 3.8.";

  input {
    leaf test-string {
      type binary;
      mandatory true;
      description
        "Input to this operation is a binary string.
         The implementation is expected to create
         a MAC over this string using the value and
         the algorithm defined as part of the
         mac-key-set.";
      reference
        "RFC 9046: Babel Information Model, Section 3.8.";
    }

    leaf mac {
      type binary;
      mandatory true;
      description
        "Input to this operation includes a MAC.
         The implementation is expected to calculate a MAC
         over the string using the value and algorithm of
```



```
        this key object and compare its calculated MAC to
        this input MAC.";
    reference
        "RFC 9046: Babel Information Model, Section 3.8.";
    }
}

output {
    leaf indication {
        type boolean;
        mandatory true;
        description
            "The output of this operation is a binary
            indication that the calculated MAC matched the
            input MAC (true) or the MACs did not match
            (false).";
        reference
            "RFC 9046: Babel Information Model, Section 3.8.";
    }
}

}

list dtls {
    key "name";

    description
        "A dtls object. If this object is implemented,
        it provides access to parameters related to the DTLS
        security mechanism.";
    reference
        "RFC 9046: Babel Information Model, Section 3.9";

    leaf name {
        type string;
        description
            "A string that uniquely identifies a dtls object.";
    }

    leaf default-apply {
        type boolean;
        mandatory true;
        description
            "A Boolean flag indicating whether this object
            instance is applied to all new interfaces, by default.
            If 'true', this instance is applied to new interfaces
            instances at the time they are created, by including it
```



```
        in the dtls-certs list under the interface. If 'false',
        this instance is not applied to new interface
        instances when they are created.";
    reference
        "RFC 9046: Babel Information Model, Section 3.9.";
}

list certs {
    key "name";

    min-elements 1;
    description
        "A set of cert objects. This contains
        both certificates for this implementation to present
        for authentication, and to accept from others.
        Certificates with a non-empty private-key
        can be presented by this implementation for
        authentication.";
    reference
        "RFC 9046: Babel Information Model, Section 3.10.";

    leaf name {
        type string;
        description
            "A unique name for this certificate that can be
            used to identify the certificate in this object
            instance, since the value is too long to be useful
            for identification. This value MUST NOT be empty
            and can only be provided when this instance is created
            (i.e., it is not subsequently writable).";
        reference
            "RFC 9046: Babel Information Model, Section 3.10.";
    }

    leaf value {
        nacm:default-deny-write;
        type string;
        mandatory true;
        description
            "The certificate in PEM format [RFC7468]. This
            value can only be provided when this instance is
            created, and is not subsequently writable.";
        reference
            "RFC 9046: Babel Information Model, Section 3.10.";
    }

    leaf type {
        nacm:default-deny-write;
```



```
    type identityref {
      base dtls-cert-types;
    }
    mandatory true;
    description
      "The certificate type of this object instance.
       The value MUST be the same as one of the
       identities listed with the base 'dtls-cert-types'.
       This value can only be provided when this
       instance is created, and is not subsequently
       writable.";
    reference
      "RFC 9046: Babel Information Model, Section 3.10.";
  }

  leaf private-key {
    nacm:default-deny-all;
    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.";
    reference
      "RFC 9046: Babel Information Model, Section 3.10.";
  }

  leaf algorithm {
    nacm:default-deny-write;
    type identityref {
      base ct:private-key-format;
    }
    mandatory true;
    description
      "Identifies the algorithm identity with which the
       private-key has been encoded. This value can only be
       provided when this instance is created, and is not
       subsequently writable.";
  }
}
}
uses routes;
}
}
}
<CODE ENDS>
```


3. IANA Considerations

This document registers a URI and a YANG module.

3.1. URI Registrations

URI: urn:ietf:params:xml:ns:yang:ietf-babel

3.2. YANG Module Name Registration

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

Name: ietf-babel

Namespace: urn:ietf:params:xml:ns:yang: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.

The security considerations outlined here are specific to the YANG data model, and do not cover security considerations of the Babel protocol or its security mechanisms in The Babel Routing Protocol [RFC8966], MAC Authentication for the Babel Routing Protocol [RFC8967], and Babel Routing Protocol over Data Transport Layer Security [RFC8968]. Each of these has its own Security Considerations section for considerations that are specific to it.

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 from a config true perspective:

'babel': This container includes an 'enable' parameter that can be used to enable or disable use of Babel on a router

'babel/constants': This container includes configuration parameters that can prevent reachability if misconfigured.

'babel/interfaces': This leaf-list has configuration parameters that can enable/disable security mechanisms and change performance characteristics of the Babel protocol. For example, enabling logging of packets and giving unintended access to the log files gives an attacker detailed knowledge of the network, and allows it to launch an attack on the traffic traversing the network device.

'babel/hmac' and 'babel/dtls': These contain security credentials that influence whether incoming packets are trusted, and whether outgoing packets are produced in a way such that the receiver will treat them as trusted.

Some of the readable data or config false nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability from a config false perspective:

'babel': Access to the information in the various nodes can disclose the network topology. Additionally, the routes used by a network device may be used to mount a subsequent attack on traffic traversing the network device.

'babel/hmac' and 'babel/dtls': These contain security credentials, including private credentials of the router; however it is required that these values not be readable.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability from a RPC operation perspective:

This model defines two actions. Resetting the statistics within an interface container would be visible to any monitoring processes, which should be designed to account for the possibility of such a reset. The "test" action allows for validation that a MAC key and MAC algorithm have been properly configured. The MAC key is a sensitive piece of information, and it is important to prevent an attacker that does not know the MAC key from being able to determine the MAC value by trying different input parameters. The "test"

action has been designed to not reveal such information directly. Such information might also be revealed indirectly, due to side channels such as the time it takes to produce a response to the action. Implementations SHOULD use a constant-time comparison between the input mac and the locally generated MAC value for comparison, in order to avoid such side channel leakage.

5. Acknowledgements

Juliusz Chroboczek provided most of the example configurations for babel that are shown in the Appendix.

6. References

6.1. Normative References

- [I-D.ietf-netconf-crypto-types]
Watsen, K., "YANG Data Types and Groupings for Cryptography", Work in Progress, Internet-Draft, draft-ietf-netconf-crypto-types-21, 14 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-netconf-crypto-types-21.txt>>.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <<https://www.rfc-editor.org/info/rfc4868>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <<https://www.rfc-editor.org/info/rfc7693>>.

- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <<https://www.rfc-editor.org/info/rfc8349>>.
- [RFC8966] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", RFC 8966, DOI 10.17487/RFC8966, January 2021, <<https://www.rfc-editor.org/info/rfc8966>>.
- [RFC8967] Do, C., Kolodziejek, W., and J. Chroboczek, "MAC Authentication for the Babel Routing Protocol", RFC 8967, DOI 10.17487/RFC8967, January 2021, <<https://www.rfc-editor.org/info/rfc8967>>.
- [RFC8968] Decimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", RFC 8968, DOI 10.17487/RFC8968, January 2021, <<https://www.rfc-editor.org/info/rfc8968>>.
- [RFC9046] Stark, B. and M. Jethanandani, "Babel Information Model", RFC 9046, DOI 10.17487/RFC9046, June 2021, <<https://www.rfc-editor.org/info/rfc9046>>.

6.2. Informative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/info/rfc2104>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Tree Diagram and Example Configurations

This section is devoted to including a complete tree diagram and examples that demonstrate how Babel can be configured.

A.1. Complete Tree Diagram

This section includes the complete tree diagram for the Babel YANG module.

```
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
+--ro seqno?              uint16
+--rw statistics-enabled?  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 metric-algorithm  identityref
|   +--rw split-horizon?    boolean
|   +--ro mcast-hello-seqno? uint16
|   +--rw mcast-hello-interval? uint16
|   +--rw update-interval?  uint16
|   +--rw mac-enable?       boolean
|   +--rw mac-key-sets*     -> ../../mac-key-set/name
|   +--rw mac-verify?       boolean
|   +--rw dtls-enable?      boolean
|   +--rw dtls-certs*       -> ../../dtls/name
|   +--rw dtls-cached-info? boolean
|   +--rw dtls-cert-prefer* -> ../../dtls/certs/type
|   +--rw packet-log-enable? boolean
|   +--ro packet-log?       inet:uri
|   +--ro statistics
|   |   +--ro discontinuity-time  yang:date-and-time
|   |   +--ro sent-mcast-hello?   yang:counter32
|   |   +--ro sent-mcast-update?  yang:counter32
|   |   +--ro sent-ucast-hello?    yang:counter32
|   |   +--ro sent-ucast-update?   yang:counter32
|   |   +--ro sent-ihu?            yang:counter32
|   |   +--ro received-packets?    yang:counter32
|   |   +---x reset
|   |   |   +---w input
|   |   |   |   +---w reset-at?  yang:date-and-time
|   |   |   +--ro output
|   |   |   |   +--ro reset-finished-at? yang:date-and-time
|   +--ro neighbor-objects* [neighbor-address]
|   |   +--ro neighbor-address  inet:ip-address
|   |   +--ro hello-mcast-history? string
|   |   +--ro hello-ucast-history? string
|   |   +--ro txcost?           int32
|   |   +--ro exp-mcast-hello-seqno? union
|   |   +--ro exp-ucast-hello-seqno? union
|   |   +--ro ucast-hello-seqno?  union
|   |   +--ro ucast-hello-interval? uint16
|   |   +--ro rxcost?            uint16
|   |   +--ro cost?             int32
+--rw mac-key-set* [name]

```



```

+--rw name string
+--rw default-apply? boolean
+--rw keys* [name]
  +--rw name string
  +--rw use-send boolean
  +--rw use-verify boolean
  +--rw value binary
  +--rw algorithm identityref
  +---x test
    +---w input
      +---w test-string binary
      +---w mac binary
    +--ro output
      +--ro indication boolean
+--rw dtls* [name]
  +--rw name string
  +--rw default-apply boolean
  +--rw certs* [name]
    +--rw name string
    +--rw value string
    +--rw type identityref
    +--rw private-key binary
    +--rw algorithm identityref
+--ro routes* [prefix]
  +--ro prefix inet:ip-prefix
  +--ro router-id? binary
  +--ro neighbor? leafref
  +--ro received-metric? union
  +--ro calculated-metric? union
  +--ro seqno? uint16
  +--ro next-hop? union
  +--ro feasible? boolean
  +--ro selected? boolean

```

A.2. Statistics Gathering Enabled

In this example, interface eth0 is being configured for routing protocol Babel, and statistics gathering is enabled. For security, HMAC-SHA256 is supported. Every sent Babel packets is signed with the key value provided, and every received Babel packet is verified with the same key value.


```
<?xml version="1.0" encoding="UTF-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
  </interface>
</interfaces>
<routing
  xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
  <control-plane-protocols>
    <control-plane-protocol>
      <type
        xmlns:babel=
          "urn:ietf:params:xml:ns:yang:ietf-babel">babel:babel</type>
      <name>name:babel</name>
      <babel
        xmlns="urn:ietf:params:xml:ns:yang:ietf-babel">
        <enable>true</enable>
        <statistics-enabled>true</statistics-enabled>
        <interfaces>
          <reference>eth0</reference>
          <metric-algorithm>two-out-of-three</metric-algorithm>
          <split-horizon>true</split-horizon>
        </interfaces>
        <mac-key-set>
          <name>hmac-sha256</name>
          <keys>
            <name>hmac-sha256-keys</name>
            <use-send>true</use-send>
            <use-verify>true</use-verify>
            <value>base64encodedvalue==</value>
            <algorithm>hmac-sha256</algorithm>
          </keys>
        </mac-key-set>
      </babel>
    </control-plane-protocol>
  </control-plane-protocols>
</routing>
```

A.3. Automatic Detection of Properties


```
<!-- In this example, babeld is configured on two interfaces
```

```
    interface eth0
    interface wlan0
```

This says to run Babel on interfaces eth0 and wlan0. Babeld will automatically detect that eth0 is wired and wlan0 is wireless, and will configure the right parameters automatically.

```
-->
```

```
<?xml version="1.0" encoding="UTF-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
  </interface>
  <interface>
    <name>wlan0</name>
    <type>ianaift:ieee80211</type>
    <enabled>true</enabled>
  </interface>
</interfaces>
<routing
  xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
  <control-plane-protocols>
    <control-plane-protocol>
      <type
        xmlns:babel=
          "urn:ietf:params:xml:ns:yang:ietf-babel">babel:babel</type>
      <name>name:babel</name>
      <babel
        xmlns="urn:ietf:params:xml:ns:yang:ietf-babel">
        <enable>true</enable>
        <interfaces>
          <reference>eth0</reference>
          <enable>true</enable>
          <metric-algorithm>two-out-of-three</metric-algorithm>
          <split-horizon>true</split-horizon>
        </interfaces>
        <interfaces>
          <reference>wlan0</reference>
          <enable>true</enable>
          <metric-algorithm>etx</metric-algorithm>
          <split-horizon>false</split-horizon>
        </interfaces>
      </babel>
```



```

    </control-plane-protocol>
  </control-plane-protocols>
</routing>

```

A.4. Override Default Properties

<!-- In this example, babeld is configured on three interfaces

```

interface eth0
interface eth1 type wireless
interface tun0 type tunnel

```

Here, interface eth1 is an Ethernet bridged to a wireless radio, so babeld's autodetection fails, and the interface type needs to be configured manually. Tunnels are not detected automatically, so this needs to be specified.

This is equivalent to the following:

```

interface eth0 metric-algorithm 2-out-of-3 split-horizon true
interface eth1 metric-algorithm etx split-horizon false
interface tun0 metric-algorithm 2-out-of-3 split-horizon true
-->

<?xml version="1.0" encoding="UTF-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
  </interface>
  <interface>
    <name>eth1</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
  </interface>
  <interface>
    <name>tun0</name>
    <type>ianaift:tunnel</type>
    <enabled>true</enabled>
  </interface>
</interfaces>
<routing
  xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
  <control-plane-protocols>
    <control-plane-protocol>
      <type

```



```

        xmlns:babel=
        "urn:ietf:params:xml:ns:yang:ietf-babel">babel:babel</type>
<name>name:babel</name>
<babel
  xmlns="urn:ietf:params:xml:ns:yang:ietf-babel">
    <enable>true</enable>
    <interfaces>
      <reference>eth0</reference>
      <enable>true</enable>
      <metric-algorithm>two-out-of-three</metric-algorithm>
      <split-horizon>true</split-horizon>
    </interfaces>
    <interfaces>
      <reference>eth1</reference>
      <enable>true</enable>
      <metric-algorithm>etx</metric-algorithm>
      <split-horizon>false</split-horizon>
    </interfaces>
    <interfaces>
      <reference>tun0</reference>
      <enable>true</enable>
      <metric-algorithm>two-out-of-three</metric-algorithm>
      <split-horizon>true</split-horizon>
    </interfaces>
  </babel>
</control-plane-protocol>
</control-plane-protocols>
</routing>

```

A.5. Configuring other Properties

<!-- In this example, two interfaces are configured for babeld

```

interface eth0
interface ppp0 hello-interval 30 update-interval 120

```

Here, ppp0 is a metered 3G link used for fallback connectivity. It runs with much higher than default time constants in order to avoid control traffic as much as possible.

-->

```

<?xml version="1.0" encoding="UTF-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>

```



```
</interface>
<interface>
  <name>ppp0</name>
  <type>ianaift:ppp</type>
  <enabled>true</enabled>
</interface>
</interfaces>
<routing
  xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
  <control-plane-protocols>
    <control-plane-protocol>
      <type
        xmlns:babel=
          "urn:ietf:params:xml:ns:yang:ietf-babel">babel:babel</type>
      <name>name:babel</name>
      <babel
        xmlns="urn:ietf:params:xml:ns:yang:ietf-babel">
        <enable>true</enable>
        <interfaces>
          <reference>eth0</reference>
          <enable>true</enable>
          <metric-algorithm>two-out-of-three</metric-algorithm>
          <split-horizon>true</split-horizon>
        </interfaces>
        <interfaces>
          <reference>ppp0</reference>
          <enable>true</enable>
          <mcast-hello-interval>30</mcast-hello-interval>
          <update-interval>120</update-interval>
          <metric-algorithm>two-out-of-three</metric-algorithm>
        </interfaces>
      </babel>
    </control-plane-protocol>
  </control-plane-protocols>
</routing>
```

Authors' Addresses

Mahesh Jethanandani
Kloud Services
California
United States of America

Email: mjethanandani@gmail.com

Barbara Stark
AT&T
Atlanta, GA
United States of America

Email: barbara.stark@att.com

Network Working Group
Internet-Draft
Updates: 6126 (if approved)
Intended status: Experimental
Expires: September 12, 2019

B. Jonglez
ENS Lyon
J. Chroboczek
IRIF, University of Paris-Diderot
March 11, 2019

Delay-based Metric Extension for the Babel Routing Protocol
draft-jonglez-babel-rtt-extension-02

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.

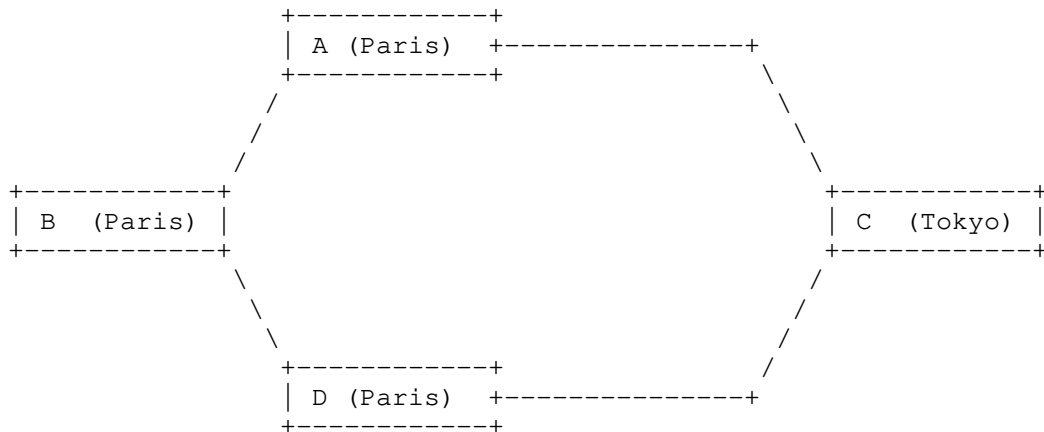
Table of Contents

1. Introduction	2
2. Protocol operation	3
2.1. Delay estimation	3
2.2. Metric computation	5
2.3. Stability issues	6
2.4. Backwards and forwards compatibility	6
3. Packet format	6
3.1. Timestamp sub-TLV in Hello TLVs	7
3.2. Timestamp sub-TLV in IHU TLVs	7
4. IANA Considerations	8
5. Security Considerations	8
6. References	8
6.1. Normative References	8
6.2. Informative References	8
Authors' Addresses	9

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

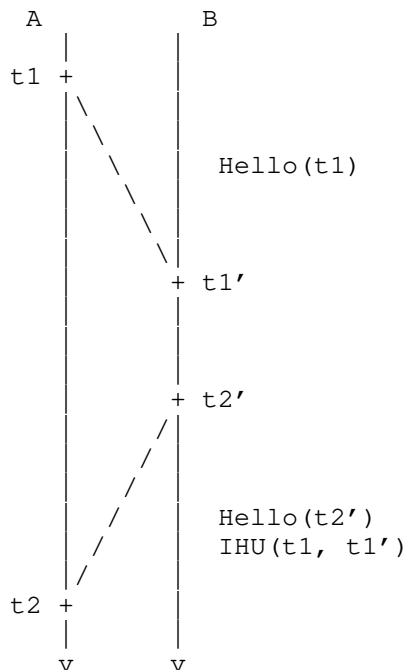
The RTT to a neighbour is estimated using an algorithm due to Mills [MILLS], originally developed for the HELLO routing protocol and later used in NTP [NTP].

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 t_1 (according to A's clock). When a node B receives A's Hello, it records in its neighbour table the timestamp t_1 as well as the time t_1' 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 t_1 and t_1' 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 t_2' (according to B's clock). Symmetrically, A will record in its neighbour table the timestamp t_2' as well as the time t_2 (according

to A's clock) at which it has received the Hello. This is illustrated in the following sequence diagram:



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.

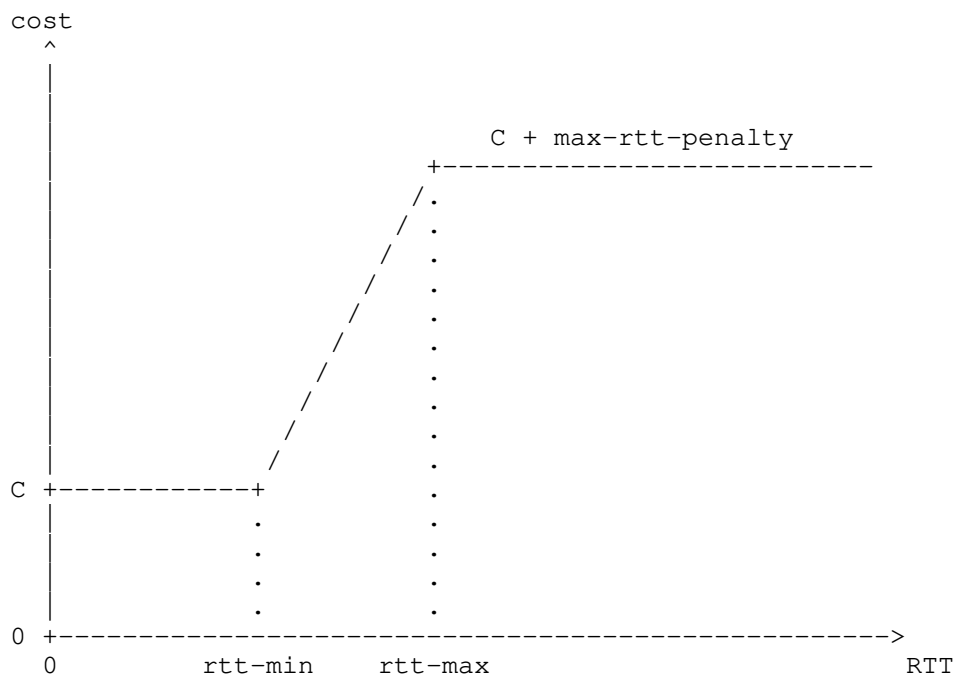
2.2. Metric computation

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`:



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:

- o the measured RTT is smoothed, which limits Babel's response to short-term RTT variations;
- o the mapping function is bounded, which avoids switching between congested routes;
- o 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	2	3	4	5	6	7	8	9
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:

Type	Name	Reference
3	Timestamp	(this document)

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

[BABEL] Chroboczek, J., "The Babel Routing Protocol", RFC 6126, February 2011.

[BABEL-EXT] Chroboczek, J., "Extension Mechanism for the Babel Routing Protocol", RFC 7557, May 2015.

6.2. Informative References

[DELAY-BASED] Jonglez, B. and J. Chroboczek, "A delay-based routing metric", March 2014.

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

- [MILLS] Mills, D., "DCN Local-Network Protocols", RFC 891, December 1983.
- [NTP] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

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