Stream: Internet Engineering Task Force (IETF)
RFC: 9616
Category: Standards Track
Published: July 2024
ISSN: 2070-1721
Authors: B. Jonglez J. Chroboczek
ENS Lyon *IRIF, Université Paris Cité*

RFC 9616
Delay-Based Metric Extension for the Babel Routing Protocol

Abstract

This document defines an extension to the Babel routing protocol that measures the round-trip time (RTT) between routers and makes it possible to prefer lower-latency links over higher-latency ones.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9616.

Copyright Notice

Table of Contents

1. Introduction

"The Babel Routing Protocol" [RFC8966] 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.

```
                 +------------+
                 | A (Paris)  +---------------+
                 +------------+                \
                /                               \
               /                                 \
              /                                   \
+------------+                                     +------------+
| B  (Paris) |                                     | C  (Tokyo) |
+------------+                                     +------------+
              \                                   /
               \                                 /
                \                               /
                 +------------+                /
                 | D (Paris)  +---------------+
                 +------------+
```

Figure 1: Four Routers in a Diamond Topology

For example, consider the topology described in Figure 1, with three routers (A, B, and D) located in Paris and a fourth router (C) 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; therefore, they will route the traffic through C in roughly half the cases.

In the first part of this document (Section 3), we specify an extension to the Babel routing protocol that produces a sequence of accurate measurements of the round-trip time (RTT) between two Babel neighbours. These measurements are not directly usable as an input to Babel's route-selection procedure since they tend to be noisy and to cause a negative feedback loop, which might give rise to frequent oscillations. In the second part (Section 4), we define an algorithm that maps the sequence of RTT samples to a link cost that can be used for route selection.

1.1. Applicability

The extension defined in Section 3 provides a sequence of accurate, but potentially noisy, RTT samples. Since the RTT is a symmetric measure of delay, this protocol is only applicable in environments where the symmetric delay is a good predictor of whether a link should be taken by routing traffic, which might not necessarily be the case in networks built over exotic link technologies.

The extension makes minimal requirements on the nodes. In particular, it does not assume synchronised clocks; it only requires that clock drift be negligible during the time interval between two Hello TLVs. Since that is on the order of a few seconds, this requirement is met even with cheap crystal oscillators, such as the ones used in consumer electronics.

The algorithm defined in Section 4 is based on a number of assumptions about the network. The assumption with the most severe consequences is that all links below a certain RTT (rtt-min in Section 4.2) can be grouped in a single category of "good" links. While this is the case in wide-area overlay networks, it makes the algorithm inapplicable in networks where distinguishing between low-latency links is important.

There are other assumptions, but they are less likely to limit the algorithm's applicability. The algorithm assumes that all links above a certain RTT (rtt-max in Section 4.2) are equally bad, and they will only be used as a last resort. In addition, in order to avoid oscillations, the algorithm is designed to react slowly to RTT variations, thus causing suboptimal routing for seconds or even minutes after an RTT change. While this is a desirable property in fixed networks, as it avoid excessive route oscillations, it might be an issue with networks with high rates of node mobility.

2. 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.

3. RTT Sampling

3.1. Data Structures

We assume that every Babel speaker maintains a local clock that counts microseconds from an arbitrary origin. We do not assume that clocks are synchronised: clocks local to distinct nodes need not share a common origin. The protocol will eventually recover if the clock is stepped, so clocks need not persist across node reboots.

Every Babel speaker maintains a Neighbour Table, described in Section 3.2.4 of [RFC8966]. This extension extends every entry in the Neighbour Table with the following data:

- the Origin Timestamp, a 32-bit timestamp (modulo 2^{32}) according to the neighbour's clock;
- the Receive Timestamp, a 32-bit timestamp (modulo 2^{32}) according to the local clock.

Both values are initially undefined.

3.2. Protocol Operation

The RTT to a neighbour is estimated using an algorithm by Mills [RFC891], originally developed for the HELLO routing protocol and later used in NTP [RFC5905].

A Babel speaker periodically sends Hello messages to its neighbours (Section 3.4.1 of [RFC8966]). Additionally, it occasionally sends a set of IHU ("I Heard You") messages, at most one per neighbour (Section 3.4.2 of [RFC8966]).

```
 A          B
   |      |
t1 +      |
   |\     |
   | \    |
   |  \   |  Hello(t1)
   |   \  |
   |    \ |
   |     \|
   |      + t1'
   |      |
   |      |             RTT = (t2 - t1) - (t2' - t1')
   |      |
   |      + t2'
   |     /|
   |    / |
   |   /  |
   |  /   |  Hello(t2')
   | /    |  IHU(t1, t1')
   |/     |
t2 +      |
   |      |
   v      v
```

Figure 2: Mills' Algorithm

In order to enable the computation of RTTs, node A **MUST** include, in every Hello that it sends, a timestamp t1 (according to A's local clock), as illustrated in Figure 2. When node B receives A's timestamped Hello, it computes the time t1' at which the Hello was received (according to B's local clock). It then **MUST** record the value t1 in the Origin Timestamp field of the Neighbour Table entry corresponding to A and the value t1' in the Receive Timestamp field of the Neighbour Table entry.

When B sends an IHU to A, it checks whether both timestamps are defined in the Neighbour Table. If that is the case, then it **MUST** ensure that its IHU TLV is sent in a packet that also contains a timestamped Hello TLV (either a normally scheduled Hello or an unscheduled Hello, see Section 3.4.1 of [RFC8966]). It **MUST** include in the IHU both the Origin Timestamp and the Receive Timestamp stored in the Neighbour Table.

Upon receiving B's packet, A computes the time t2 (according to its local clock) at which it was received. Then, A **MUST** verify that it contains both a Hello TLV with timestamp t2' and an IHU TLV with two timestamps: t1 and t1'. If that is the case, A computes the value:

```
RTT = (t2 - t1) - (t2' - t1')
```

(where all computations are done modulo 2^{32}), which is a measurement of the RTT between A and B. (A then stores the values t2' and t2 in its Neighbour Table, as B did before.)

This algorithm has a number of desirable properties:

1. The algorithm is symmetric. A and B use the same procedures for timestamping packets and computing RTT samples; both nodes produce one RTT sample for each received (Hello, IHU) pair.
2. Since there is no requirement that t1' and t2' be equal, the protocol is asynchronous: the only change to Babel's message scheduling is the requirement that a packet containing an IHU also contain a Hello.
3. Since the algorithm only ever computes differences of timestamps according to a single clock, it does not require synchronised clocks.
4. The algorithm requires very little additional state: a node only needs to store the two timestamps associated with the last hello received from each neighbour.
5. Since the algorithm only requires piggybacking one or two timestamps on each Hello and IHU TLV, it makes efficient use of network resources.

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

In order for RTT values to be consistent between implementations, timestamps need to be computed at roughly the same point in the network stack. Transmit timestamps **SHOULD** be computed just before the packet is passed to the network stack (i.e., before it is subjected to any queueing delays); receive timestamps **SHOULD** be computed just after the packet is received from the network stack.

3.3. Wrap-Around and Node Restart

Timestamp values are a count of microseconds stored as a 32-bit unsigned integer; thus, they wrap around every 71 minutes or so. What is more, a node may occasionally reboot and restart its clock at an arbitrary origin. For these reasons, very old timestamps or nonsensical timestamps **MUST NOT** be used to yield RTT samples.

The following algorithm can be used to discard obsolete samples. When a node receives a packet containing a Hello and an IHU, it compares the current local time t2 with the Origin Timestamp contained in the IHU; if the Origin Timestamp appears to be in the future, or if it is in the past by more than a time T (the value T = 3 minutes is recommended), then the timestamps are still recorded in the Neighbour Table, but they are not used for computation of an RTT sample.

Similarly, the node compares the Hello's timestamp with the Receive Timestamp recorded in the Neighbour Table; if the Hello's timestamp appears to be older than the recorded timestamp, or if it appears to be more recent by an interval larger than the value T, then the timestamps are not used for computation of an RTT sample.

3.4. Implementation Notes

The accuracy of the computed RTT samples depends on Transmit Timestamps being computed as late as possible before a packet containing a Hello TLV is passed to the network stack, and Receive Timestamps being computed as early as possible after reception of a packet containing a (Hello, IHU) pair. We have found the following implementation strategy to be useful.

When a Hello TLV is buffered for transmission, we insert a PadN sub-TLV (Section 4.7.2 of [RFC8966]) with a length of 4 octets within the TLV. When the packet is ready to be sent, we check whether it contains a 4-octet PadN sub-TLV; if that's the case, we overwrite the PadN sub-TLV with a Timestamp sub-TLV with the current time, and send out the packet.

Conversely, when a packet is received, we immediately compute the current time and record it with the received packet. We then process the packet as usual and use the recorded timestamp in order to compute an RTT sample.

The protocol is designed to survive the clock being reset when a node reboots; on POSIX systems, this makes it possible to use the CLOCK_MONOTONIC clock for computing timestamps. If CLOCK_MONOTONIC is not available, CLOCK_REALTIME may be used, since the protocol is able to survive the clock being occasionally stepped.

4. RTT-Based Route Selection

The protocol described above yields a series of RTT samples. While these samples are fairly accurate, they are not directly usable as an input to the route-selection procedure, for at least three reasons:

1. In the presence of bursty traffic, routers experience transient congestion, which causes occasional spikes in the measured RTT. Thus, the RTT signal may be noisy and require smoothing before it can be used for route selection.
2. Using the RTT signal for route selection gives rise to a negative feedback loop. When a route has a low RTT, it is deemed to be more desirable: this causes it to be used for more data traffic, which may lead to congestion, which in turn increases the RTT. Without some form of hysteresis, using RTT for route selection would lead to oscillations between parallel routes, which would lead to packet reordering and negatively affect upper-layer protocols (such as TCP).
3. Even in the absence of congestion, the RTT tends to exhibit some variation. If the RTTs of two parallel routes oscillate around a common value, using the RTT as input to route selection will cause frequent routing oscillations, which, again, indicates the need for some form of hysteresis.

In this section, we describe an algorithm that integrates smoothing and hysteresis. It has also been shown to behave well both in simulation and experimentally over the Internet [DELAY-BASED] and is **RECOMMENDED** when RTT information is being used for route selection. The algorithm is structured as follows:

- the RTT values are first smoothed in order to avoid instabilities due to outliers (Section 4.1);
- the resulting smoothed samples are mapped to a cost using a bounded, non-linear mapping, which avoids instabilities at the lower and upper end of the RTT range (Section 4.2);
- a hysteresis filter is applied in order to limit the amount of oscillation in the middle of the RTT range (Section 4.3).

4.1. Smoothing

The RTT samples provided by Mills' algorithm are fairly accurate, but noisy: experiments indicate the occasional presence of individual samples that are much larger than the expected value. Thus, some form of smoothing **SHOULD** be applied in order to avoid instabilities due to occasional outliers.

An implementation **MAY** use the exponential average algorithm, which is simple to implement and appears to yield good results in practice [DELAY-BASED]. The algorithm is parameterised by a constant α, where $0 < \alpha < 1$, which controls the amount of smoothing being applied. For each neighbour, it maintains a smoothed value RTT, which is initially undefined. When the first sample RTT0 is measured, the smoothed value is set to the value of RTT0. At each new sample RTTn, the smoothed value is set to a weighted average of the previous smoothed value and the new sample:

```
RTT := α RTT + (1 - α) RTTn
```

The smoothing constant α **SHOULD** be between 0.8 and 0.9; the value 0.836 is the **RECOMMENDED** default.

4.2. Cost Computation

The smoothed RTT value obtained in the previous step needs to be mapped to a link cost, suitable for input to the metric computation procedure (Section 3.5.2 of [RFC8966]). Obviously, the mapping should be monotonic (larger RTTs imply larger costs). In addition, the mapping should be constant beyond a certain value (all very bad links are equally bad) so that congested links do not contribute to routing instability. The mapping should also be constant around 0; this is so small oscillations in the RTT of low-RTT links do not contribute to routing instability.

```
cost
  ^
  |
  |
  |                           C + max-rtt-penalty
  |                       +---------------------------
  |                      /.
  |                     / .
  |                    /  .
  |                   /   .
  |                  /    .
  |                 /     .
  |                /      .
  |               /       .
  |              /        .
  |             /         .
C +------------+          .
  |            .          .
  |            .          .
  |            .          .
  |            .          .
0 +---------------------------------------------------->
  0         rtt-min    rtt-max                          RTT
```

Figure 3: Mapping from RTT to Link Cost

Implementations **SHOULD** use the mapping described in Figure 3, which is parameterised by three parameters: rtt-min, rtt-max, and max-rtt-penalty. For RTT values below rtt-min, the link cost is just the nominal cost C of a single hop. 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.

The value rtt-min should be slightly larger than the RTT of a local, uncongested link. The value rtt-max should be the RTT above which a link should be avoided if possible, either because it is a long-distance link or because it is congested; reducing the value of rtt-max improves stability, but prevents the protocol from discriminating between high-latency links. As for max-rtt-penalty, it controls how much the protocol will penalise long-distance links. The default values rtt-min = 10 ms, rtt-max = 120 ms, and max-rtt-penalty = 150 are **RECOMMENDED**.

4.3. Hysteresis

Even after applying a bounded mapping from smoothed RTT to a cost value, the cost may fluctuate when a link's RTT is between rtt-min and rtt-max. Implementations **SHOULD** use a robust hysteresis algorithm, such as the one described in Appendix A.3 of [RFC8966].

5. 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 Appendix D of [RFC8966], 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; 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 protocol to extend the packet format with additional data, for example, high-precision or absolute timestamps.

6. Packet Format

This extension defines the Timestamp sub-TLV whose Type field has a value of 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 (modulo 2^{32}), expressed in units of one microsecond, counting from an arbitrary origin. Timestamps wrap around every 4295 seconds, or roughly 71 minutes (see also Section 3.3).

6.1. 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       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          (continued)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type: Set to 3 to indicate a Timestamp sub-TLV.

Length: The length of the body in octets, 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 (the first 4 octets are interpreted as described above) and any extra data contained in this sub-TLV **MUST** be silently ignored. If the Length field is smaller than the expected 4 octets, then this sub-TLV **MUST** be ignored (and the remainder of the enclosing TLV processed as usual).

6.2. Timestamp Sub-TLV in IHU TLVs

When contained in an IHU 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     |        Origin Timestamp       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          (continued)          |        Receive Timestamp      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          (continued)          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type: Set to 3 to indicate a Timestamp sub-TLV.

Length: The length of the body in octets, 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 the node to which the enclosing IHU TLV applies.

Receive Timestamp: The time, according to the sender's clock, at which the last timestamped Hello TLV was received from the node to which the enclosing IHU TLV applies.

If the Length field is larger than the expected 8 octets, the sub-TLV **MUST** be processed normally (the first 8 octets are interpreted as described above), and any extra data contained in this sub-TLV **MUST** be silently ignored. If the Length field is smaller than the expected 8 octets, then this sub-TLV **MUST** be ignored (and the remainder of the enclosing TLV processed as usual).

7. IANA Considerations

IANA has added the following entry to the "Babel Sub-TLV Types" registry:

Type	Name	Reference
3	Timestamp	RFC 9616

Table 1

8. Security Considerations

This extension adds timestamping data to two of the TLVs sent by a Babel router. By broadcasting the value of a reasonably accurate local clock, these additional data might make a node more susceptible to timing attacks.

Broadcasting an accurate time raises privacy issues. The timestamps used by this protocol have an arbitrary origin; therefore, they do not leak a node's boot time or time zone. However, having access to accurate timestamps could allow an attacker to determine the physical location of a node. Nodes might avoid disclosure of location information by not including Timestamp sub-TLVs in the TLVs that they send, which will cause their neighbours to fall back to hop-count routing.

9. References

9.1. Normative References

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

[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>.

9.2. Informative References

[DELAY-BASED] Jonglez, B., Boutier, M., and J. Chroboczek, "A delay-based routing metric", DOI 10.48550/arXiv.1403.3488, March 2014, <http://arxiv.org/abs/1403.3488>.

[RFC891] Mills, D., "DCN Local-Network Protocols", STD 44, RFC 891, DOI 10.17487/RFC0891, December 1983, <https://www.rfc-editor.org/info/rfc891>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

Acknowledgements

The authors are indebted to Jean-Paul Smets, who prompted the investigation that originally lead to this protocol. We are also grateful to Donald Eastlake, 3rd, Toke Høiland-Jørgensen, Maria Matejka, David Schinazi, Pacal Thubert, Steffen Vogel, and Ondřej Zajiček.

Authors' Addresses

Baptiste Jonglez
ENS Lyon
France
Email: baptiste.jonglez@ens-lyon.org

Juliusz Chroboczek
IRIF, Université Paris Cité
Case 7014
75205 Paris Cedex 13
France
Email: jch@irif.fr