Workgroup: Network Working Group
Updates: 8967 (if approved)
Published: 26 July 2023
Intended Status: Standards Track
Expires: 27 January 2024
Authors: B. Jonglez ENS Lyon
J. Chroboczek IRIF, Université Paris Cité

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 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 27 January 2024.

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.

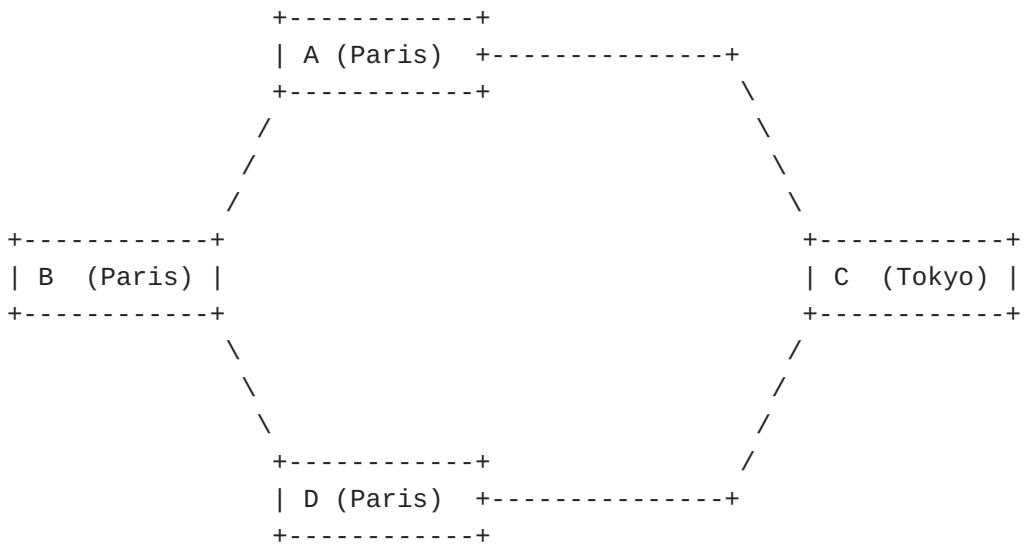


Figure 1: Four routers in a diamond topology

Consider for example 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, 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 4).

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 fine granularity of the timestamps used (1µs) should make it possible to experiment with RTT-based metrics on this kind of link layers.

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 according to the local clock.

Both values are initially undefined.

3.2. Protocol operation

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 Hello messages to its neighbours (Section 3.4.1 of [RFC8966]). Additionally, it occasionally sends a set of IHU messages, at most one per neighbour (Section 3.4.2 of [RFC8966]).

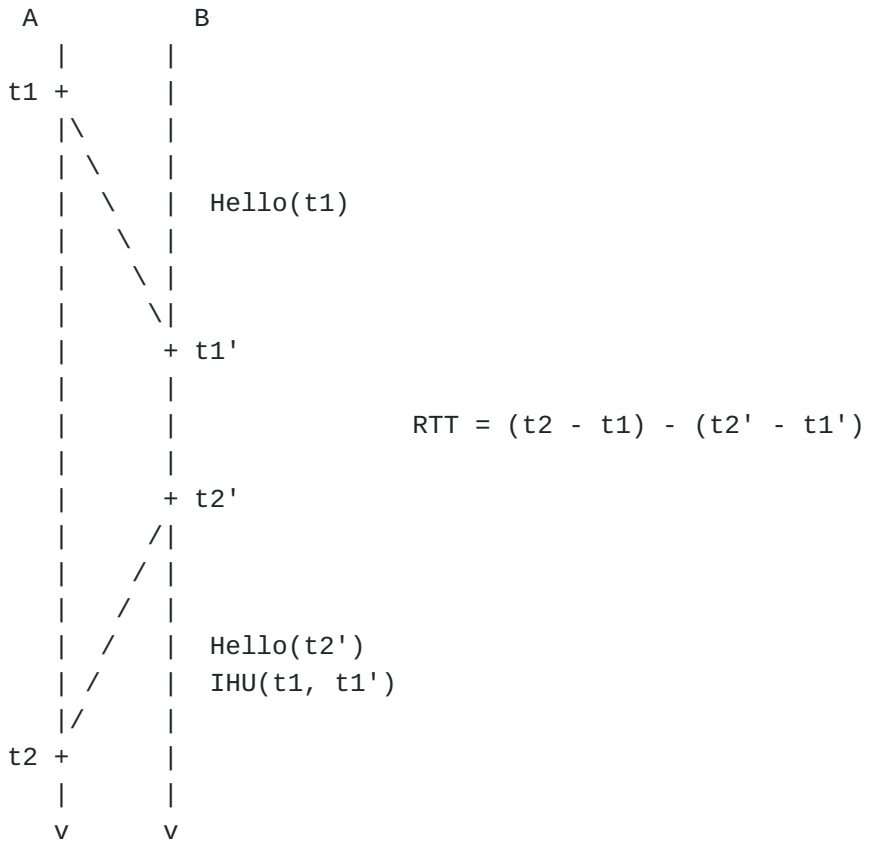


Figure 2: Mill's algorithm

In order to enable the computation of RTTs, a 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 a 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. A MUST then 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. First, the algorithm is symmetric: A and B use the same procedures for timestamping packets and computing RTT samples, and both nodes produce one RTT sample for each received (Hello, IHU) pair. Second, 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. Third, since it only ever computes differences of timestamps according to a single clock, it does not require synchronised clocks. Fourth, 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 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 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.

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.

We suggest the following algorithm to achieve that. When a node receives a packet containing a Hello and an IHU, it SHOULD compare 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 SHOULD NOT be used for computation of an RTT sample.

Similary, 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 SHOULD NOT be 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 transmission of a packet containing a Hello TLV, 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; we then 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.

First of all, 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 requires smoothing before it can be used for route selection.

Second, 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, which 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 might lead to packet reordering and negatively affect upper-layer protocols (such as TCP).

Third, 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 and has been shown to behave well both in simulation and experimentally over the Internet [DELAY-BASED]. While implementations MAY use this algorithm, it is considered experimental, since we do not currently have a theoretical understanding of its behaviour, and in particular we do not know by how much it could be simplified without impairing its functionality.

4.1. Smoothing

The RTT samples provided by Mills' algorithm are fairly accurate, but somewhat noisy: individual samples may be outliers and indicate a value much larger than the correct one. Thus, some smoothing needs to be applied first, in order to get rid of these outliers.

Our current implementation uses an exponential average, as described in [DELAY-BASED]. Other algorithms have also been considered, such as a moving average or a moving minimum, but we have not evaluated their behaviour.

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, in order to enhance stability, the mapping should be bounded: above a certain RTT, all links are equally bad, and therefore congested links do not contribute to routing instability.

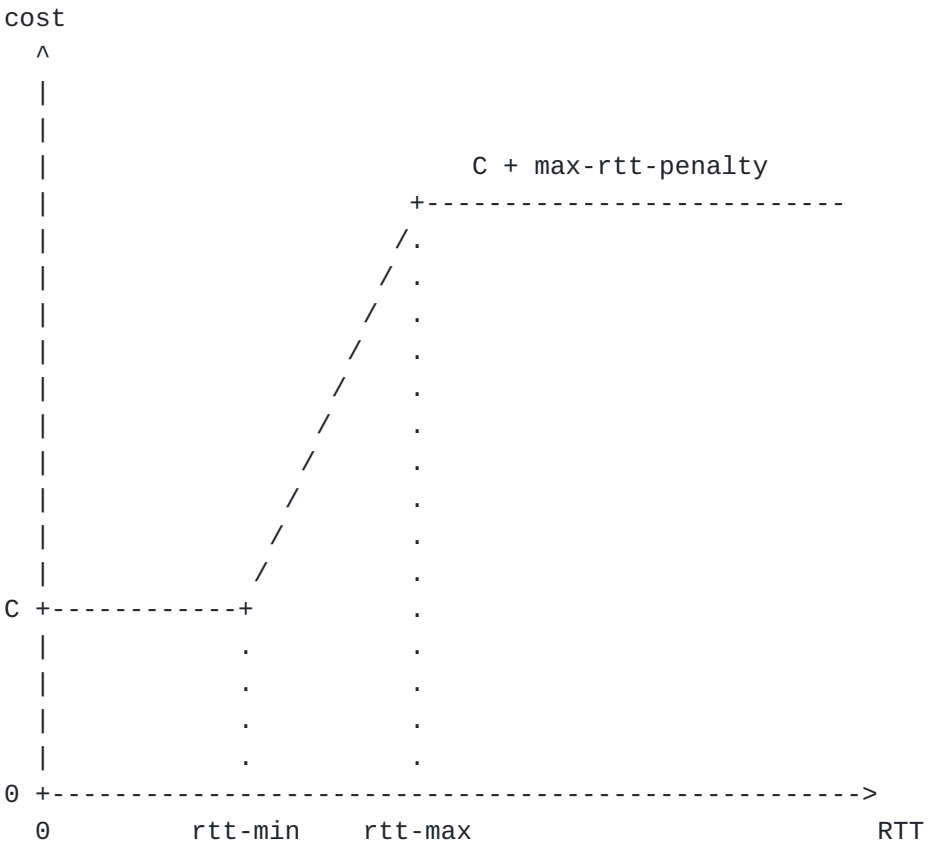


Figure 3: Mapping from RTT to link cost

We currently use the function described in Figure 3 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 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; making rtt-max smaller improves stability, but prevents the protocol from discriminating between high-latency links. As to max-rtt-penalty, it controls how much the protocol will penalise long-distance links. We suggest the default values rtt-min = 10ms, rtt-max = 120ms, max-rtt-penalty = 150.

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. This is effectively mitigated by using 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, 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 protocol to extend the packet format with additional data, for example high-precision or absolute timestamps.

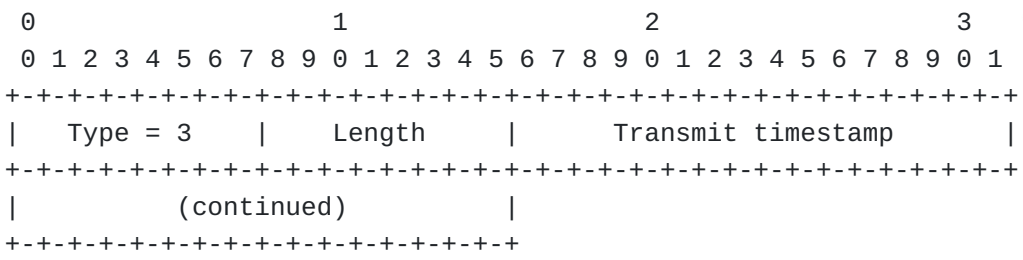
6. Packet format

This extension defines the Timestamp sub-TLV whose Type field has 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 (modulo 2^32), expressed in units of one microsecond, counting from an arbitrary origin. Timestamps wrap around every 4295 seconds, or rougly 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:



Fields:

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

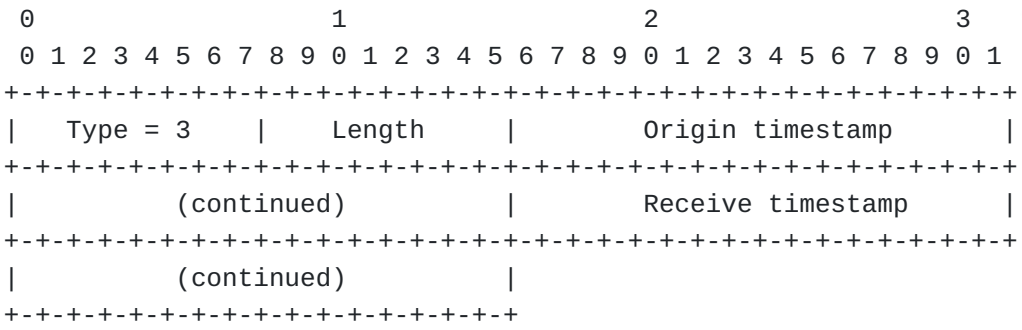
Length The length of the body in octets, exclusive of the Type and Length fields.

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 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:



Fields:

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 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	(this document)

Table 1

8. Security Considerations

This extension adds additional timestamping data to two of the TLVs sent by a Babel router. Since the timestamps use an arbitrary origin, they do not leak private data, such as a node's timezone. However, by broadcasting the value of a reasonably accurate local clock, they might make a node more susceptible to timing attacks.

9. Acknowledgements

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

10. References

10.1. Normative References

[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>.

[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/rfc/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/rfc/rfc8174>.

10.2. 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, <https://www.rfc-editor.org/rfc/rfc891>.

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

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