Delay-based routing for the Babel protocol
`draft-jonglez-babel-rtt-extension-02`

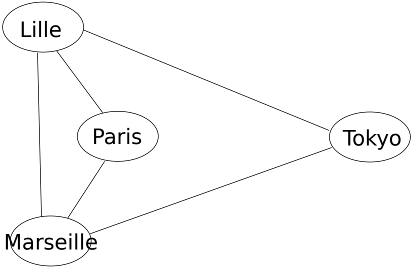
Juliusz Chroboczek
Joint work with Baptise Jonglez

28 March 2019

Babel

Problem statement

Nexedi have been running a global overlay network between datacenters:



What happens when the Lille-Marseille link is down?

In 1/2 of the cases, unextended Babel chose to reroute the traffic through Tokyo.

Nexedi were not happy.

Solution: use RTT

In 1/2 of the cases, unextended Babel chooses to reroute the traffic through Tokyo.
That's not good.

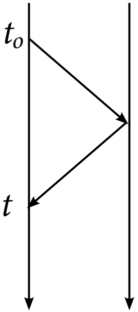
Initial suggestion: a GPS in every data center.
That's reportedly not practical.

Idea: measure RTT (two-way delay) and derive a metric from that. But

- the natural way to measure RTT requires asymmetric, synchronous interaction; Babel is a symmetric, asynchronous protocol;
- using RTT as input to a routing metric causes a (negative) feedback loop, which may lead to oscillations.

Measuring RTT (1)

The naive algorithm



The natural way to measure RTT is asymmetric and synchronous.

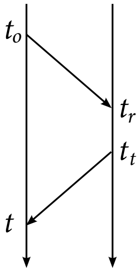
Client says “ping!”.
Server replies “pong!” as fast as possible.

$\text{RTT} = t - t_o$.

Babel is a symmetric, asynchronous algorithm.
The naive “ping” algorithm is a poor fit for Babel.

Measuring RTT (2)

Mills' algorithm



Mills' algorithm, used in HELLO and NTP.

The remote peer sends a packet with:

- t_o, the origin timestamp;
- t_r, the reference timestamp;
- t_t, the transmit timestamp.

$\mathrm{RTT} = (t - t_o) - (t_t - t_r)$.

This is a symmetric, asynchronous algorithm that doesn't require clocks to be synchronised.

Its accuracy depends on:

- t_t computed as late as possible before transmission;
- t computed as early as possible after reception;
- clock drift negligible during a packet exchange.

Encoding Mills' algorithm in Babel

Timestamps stored in sub-TLVs:

- transmission timestamp t_t stored in Hello TLV: that's a property of the packet;
- origin and reference timestamp in IHU TLV: that's a property of the neighbour.

Granularity of timestamp is $1\,\mu s$.
(Originally 10 ms, but Dave complained.)

Packet format

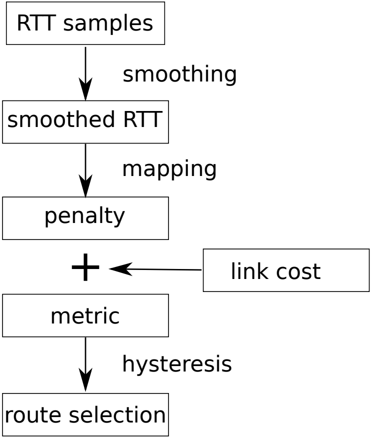
Timestamp in Hello:

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

Timestamp in IHU:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |    Length     |        Origin timestamp       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |        Receive timestamp      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

From RTT to route selection



Mills' algorithm yields RTT samples.

Our goal is route selection.

The RTT samples are processed in order to minimise:

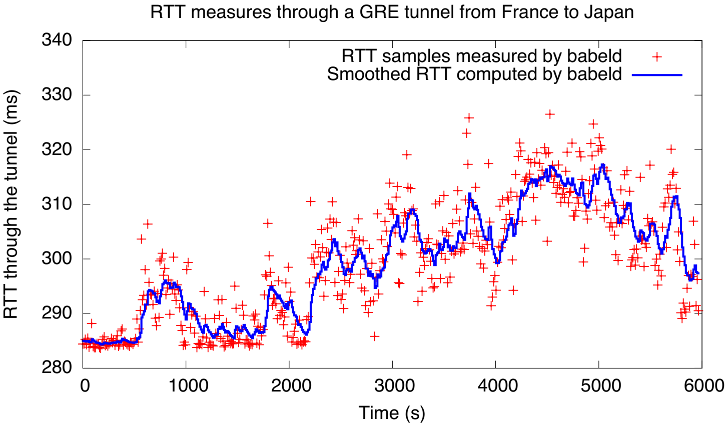
- noisy signal;
- oscillations

Smoothing

The RTT samples are smoothed in order to avoid using a noisy signal. We use TCP's exponential average.

At each sample RTT_n,

$$\mathrm{RTT} := \alpha \cdot \mathrm{RTT} + (1-\alpha)\cdot \mathrm{RTT}_n \qquad (\alpha = 0.836)$$



Feedback loop

Using a metric for RTT causes a feedback loop:

- we direct data to links with low RTT;
- which causes their RTT to increase.

This feedback loop causes persistent oscillations.

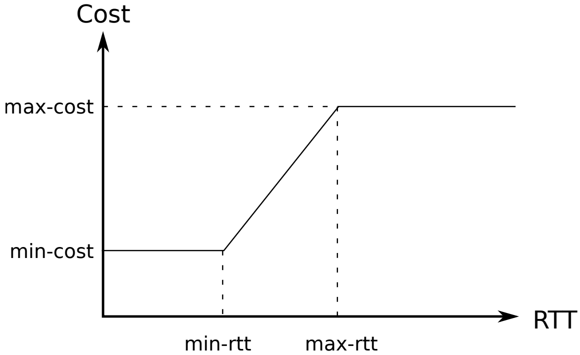
In principle, Babel doesn't care: even in the presence of oscillations, it pushes packets towards the destination according to loop-free paths. However, oscillations cause reordering, and higher-layer protocols do care.

Some mechanism is needed to limit the frequency of oscillations.

Cost computation

Saturation

Smoothed RTT is mapped to a cost using a saturating function:



The value of max-rtt is chosen so that congested links are in the saturated state. In effect, we no longer oscilate between saturated links.

This requires manual tuning.

Hysteresis

Saturation avoids oscillating between congested links. Still, using RTT might cause us to switch between links with very similar RTT.

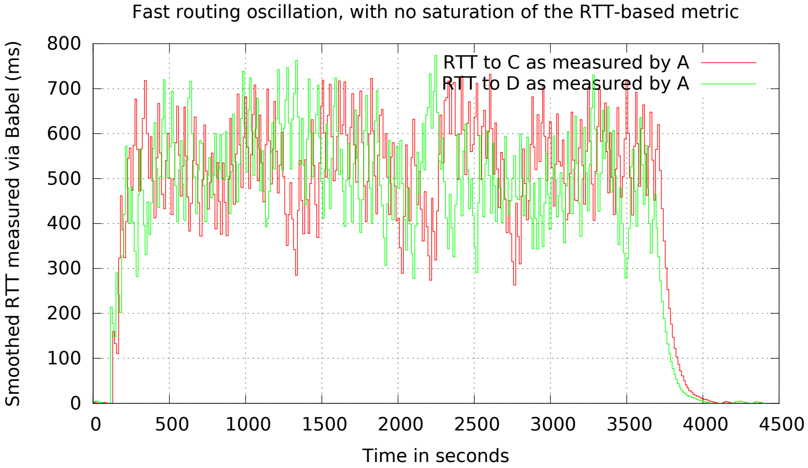
Solution: apply hysteresis before route selection.

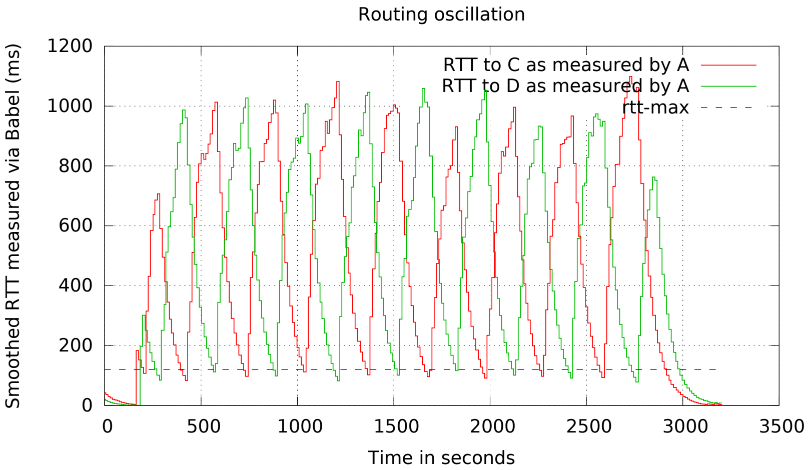
Let M_a be the announced metric. We compute a smoothed metric M_s

$$M_s := \beta(\delta) \cdot M_s + (1 - \beta(\delta)) \cdot M_a$$

- M_a is a short-term metric;
- M_s is a long-term metric.

We only switch routes when both are better.


Fast routing oscillation, with no saturation of the RTT-based metric
Smoothed RTT measured via Babel (ms)
800
700
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Time in seconds
RTT to C as measured by A
RTT to D as measured by A


Routing oscillation
Smoothed RTT measured via Babel (ms)
Time in seconds
RTT to C as measured by A
RTT to D as measured by A
rtt-max
0
200
400
600
800
1000
1200
0
500
1000
1500
2000
2500
3000
3500

Open issues

This extension has been used for years in production, with excellent results. Still, open issues remain.

Packet format:

- an IHU sub-TLV can only be interpreted if the packet contains a Hello sub-TLV.

(Is this a problem? The alternative, including a transmit timestamp in each IHU, requires duplicating the timestamp multiple times.)

Algorithms:

- we didn’t evaluate different smoothing functions;
- bad things happen if max-rtt is too large;
- we didn’t work on auto-tuning max-rtt;
- we didn’t evaluate the effect of hysteresis;

Conclusion

`draft-jonglez-babel-rtt-extension-02` describes a simple protocol extension that has been used in production for years, with excellent results.

It is used together with a bunch of algorithms that are not fully understood, but happen to work well.

Suggestion:

- standardise the packet format;
- describe the algorithms in an informative appendix;
- do more research on the algorithms.