LEDBAT WG	S. Shalunov
Internet-Draft	BitTorrent Inc
Intended status: Experimental	March 04, 2009
Expires: September 5, 2009	

TOC

Low Extra Delay Background Transport (LEDBAT)
draft-shalunov-ledbat-congestion-00.txt

Status of this Memo

Copyright Notice

Abstract

LEDBAT is an alternative experimental congestion control algorithm. LEDBAT enables an advanced networking application to minimize the extra delay it induces in the bottleneck while saturating the bottleneck. It thus implements an end-to-end version of scavenger service. LEDBAT has been been implemented in BitTorrent DNA, as the exclusive congestion control mechanism, and in uTorrent, as an experimental mechanism, and deployed in the wild with favorable results.

Table of Contents

---

1. Requirements notation

TOC

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] (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.).

---

2. Introduction

TOC

The standard congestion control in TCP is based on loss and has not been designed to drive delay to any given value. Because TCP needs losses to back off, when a FIFO bottleneck lacks AQM, TCP fills the buffer, effectively maximizing possible delay. Large number of the thinnest links in the Internet, particularly most uplinks of home

connections, lack AQM. They also frequently contain enough buffer space to get delays into hundreds of milliseconds and even seconds. There is no benefit to having delays this large, but there are very substantial drawbacks for interactive applications: games and VoIP become impossible and even web browsing becomes very slow.

While a number of delay-based congestion control mechanisms have been proposed, they were generally not designed to minimize the delay induced in the network.

LEDBAT is designed to allow to keep the latency across the congested bottleneck low even as it is saturated. This allows applications that send large amounts of data, particularly upstream on home connections, such as peer-to-peer application, to operate without destroying the user experience in interactive applications. LEDBAT takes advantage of delay measurements and backs off before loss occurs. It has been deployed by BitTorrent in the wild with the BitTorrent DNA client and now, experimentally, with the uTorrent client. This mechanism not only allows to keep delay across a bottleneck low, but also yields quickly in the presence of competing traffic with loss-based congestion control.

Beyond its utility for P2P, LEDBAT enables other advanced networking applications to better get out of the way of interactive apps.

In addition to direct and immediate benefits for P2P and other application that can benefit from scavenger service, LEDBAT could point the way for a possible future evolution of the Internet where loss is not part of the designed behavior and delay is minimized.

---

3. LEDBAT design goals

TOC

LEDBAT design goals are:

1. saturate the bottleneck
2. keep delay low when no other traffic is present
3. quickly yield to traffic sharing the same bottleneck queue that uses standard TCP congestion control
4. add little to the queuing delays induced by TCP traffic
5. operate well in networks with FIFO queuing with drop-tail discipline
6. be deployable for popular applications that currently comprise noticeable fractions of Internet traffic

7. where available, use explicit congestion notification (ECN), active queue management (AQM), and/or end-to-end differentiated services (DiffServ).

---

4. LEDBAT motivation

TOC

This section describes LEDBAT informally and provides some motivation. It is expected to be helpful for general understanding and useful in discussion of the properties of LEDBAT.
Without a loss of generality, we can consider only one direction of the data transfer. The opposite direction can be treated identically.

4.1. Simplest network topology

TOC

Consider first the desired behavior when there's only a single bottleneck and no competing traffic whatsoever, not even other LEDBAT connections. The design goals obviously need to be robustly met for this trivial case.

4.2. Extra delay

TOC

Consider the queuing delay on the bottleneck. This delay is the extra delay induced by congestion control. One of our design goals is to keep this delay low. However, when this delay is zero, the queue is empty and so no data is being transmitted and the link is thus not saturated. Hence, our design goal is to keep the queuing delay low, but non-zero.

4.3. Queuing delay target

TOC

How low do we want the queuing delay to be? Because another design goal is to be deployable on networks with only simple FIFO queuing and drop-tail discipline, we can't rely on explicit signaling for the queuing delay. So we're going to estimate it using external measurements. The external measurements will have an error at least on the order of best-case scheduling delays in the OSes. There's thus a good reason to try to make the queuing delay larger than this error. There's no reason that would want us to push the delay much further up. Thus, we will have a delay target that we would want to maintain.

4.4. Need to measure delay

TOC

To maintain delay near the target, we have to use delay measurements. Lacking delay measurements, we'd have to go only by loss (when ECN is lacking). For loss to occur (on a FIFO link with drop-tail discipline), the buffer must first be filled. This would drive the delay to the largest possible value for this link, thus violating our design goal of keeping delay low.

4.5. Queing delay estimate

TOC

Since our goal is to control the queuing delay, it is natural to maintain an estimate of it. Let's call delay components propagation, serialization, processing, and queuing. All components but queuing are nearly constant and queuing is variable. Because queuing delay is always positive, the constant propagation+serialization+processing delay is no less than the minimum delay observed. Assuming that the queuing delay distribution density has non-zero integral from zero to any sufficiently small upper limit, minimum is also an asymptotically consistent estimate of the constant fraction of the delay. We can thus estimate the queuing delay as the difference between current and base delay as usual.

4.6. Controller

TOC

When our estimate of the queuing delay is lower than the target, it's natural to send faster. When our estimate is higher, it's natural to send slower. To avoid trivial oscillations on round-trip-time (RTT) scale, the response of the controller needs to be near zero when the estimate is near the target. To converge faster, the response needs to increase as the difference increases. The simplest controller with this property is the linear controller, where the response is proportional to the difference between the estimate and the target. This controller happens to work well in practice obviating the need for more complex ones.

TOC

4.7. Max rampup rate same as TCP

The maximum speed with which we can increase our congestion window is then when queuing delay estimate is zero. To be on the safe side, we'll make this speed equal to how fast TCP increases its sending speed. Since queuing delay estimate is always non-negative, this will ensure never ramping up faster than TCP would.

4.8. Halve on loss

TOC

Further, to deal with severe congestion when most packets are lost and to provide a safety net against incorrect queuing delay estimates, we'll halve the window when a loss event is detected. We'll do so at most once per RTT.

4.9. Yield to TCP

TOC

Consider competition between a LEDBAT connection and a connection governed by loss-based congestion control (on a FIFO bottleneck with drop-tail discipline). Loss-based connection will need to experience loss to back off. Loss will only occur after the connection experiences maximum possible delays. LEDBAT will thus receive congestion indication sooner than the loss-based connection. If LEDBAT can ramp down faster than the loss-based connection ramps up, LEDBAT will yield. LEDBAT ramps down when queuing delay estimate exceeds the target: the more the excess, the faster the ramp-down. When the loss-based connection is standard TCP, LEDBAT will yield at precisely the same rate as TCP is ramping up when the queuing delay is double the target.

4.10. Need for one-way delay

TOC

Now consider a case when one link direction is saturated with unrelated TCP traffic while another direction is near-empty. Consider LEDBAT sending in the near-empty direction. Our design goal is to saturate it. However, if we pay attention to round-trip delays, we'll sense the delays on the reverse path and respond to them as described in the previous paragraph. We must, thus, measure one-way delay and use that for our queuing delay estimate.

TOC

4.11. Measuring one-way delay

A special IETF protocol, One-Way Active Measurement Protocol (OWAMP), exists for measuring one-way delay. However, since LEDBAT will already be sending data, it is more efficient to add a timestamp to the packets on the data direction and a measurement result field on the acknowledgement direction. This also prevents the danger of measurement packets being treated differently from the data packets. The failure case would be better treatment of measurement packets, where the data connection would be driven to losses.

4.12. Route changes

TOC

Routes can change. To deal, base delay needs to be computed over a period of last few minutes instead of since the start of connection. The tradeoff is: for longer intervals, base is more accurate; for shorter intervals, reaction to route changes is faster.
A convenient way to implement an approximate minimum over last N minutes is to keep separate minima for last N+1 minutes (last one for the partial current minute).

4.13. Timestamp errors

TOC

One-way delay measurement needs to deal with timestamp errors. We'll use the same locally linear clock model as Network Time Protocol (NTP). This model is valid for any differentiable clocks. The clock will thus have a fixed offset from the true time and a skew. We'll consider what we need to do about the offset and the skew separately.

4.13.1. Clock offset

TOC

First, consider the case of zero skew. The offset of each of the two clocks shows up as a fixed error in one-way delay measurement. The difference of the offsets is the absolute error of the one-way delay estimate. We won't use this estimate directly, however. We'll use the difference between that and a base delay. Because the error (difference of clock offsets) is the same for the current and base delay, it cancels from the queuing delay estimate, which is what we'll use. Clock offset is thus irrelevant to the design.

4.13.2. Clock skew

TOC

Now consider the skew. For a given clock, skew manifests in a linearly changing error in the time estimate. For a given pair of clocks, the difference in skews is the skew of the one-way delay estimate. Unlike the offset, this no longer cancels in the computation of the queuing delay estimate. On the other hand, while the offset could be huge, with some clocks off by minutes or even hours or more, the skew is typically not too bad. For example, NTP is designed to work with most clocks, yet it gives up when the skew is more than 500 parts per million (PPM). Typical skews of clocks that have never been trained seem to often be around 100-200 PPM. Previously trained clocks could have 10-20 PPM skew due to temperature changes. A 100-PPM skew means accumulating 6 milliseconds of error per minute. The expiration of base delay related to route changes mostly takes care of clock skew. A technique to specifically compute and cancel it is trivially possible and involves tracking base delay skew over a number of minutes and then correcting for it, but usually isn't necessary, unless the target is unusually low, the skew is unusually high, or the base interval is unusually long. It is not further described in this document.

4.14. Noise filtering

TOC

In addition to timestamp errors, one-way delay estimate includes an error of measurement when part of the time measured was spent inside the sending or the receiving machines. Different views are possible on the nature of this delay: one view holds that, to the extent this delay internal to a machine is not constant, it is a variety of queuing delay and nothing needs to be done to detect or eliminate it; another view holds that, since this delay does not have the same characteristics as queuing delay induced by a fixed-capacity bottleneck, it is more correctly classified as non-constant processing delay and should be filtered out. In practice, this doesn't seem to matter very much one way or the other. The way to filter the noise out is to observe, again, that the noise is always nonnegative and so a good filter is the minimum of several recent delay measurements.

4.15. Safety of LEDBAT

TOC

LEDBAT is most aggressive when its queuing delay estimate is most wrong and is as low as it can be. Queuing delay estimate is nonnegative, therefore the worst possible case is when somehow the estimate is always returned as zero. In this case, LEDBAT will ramp up as fast as

TCP and halve the rate on loss. Thus, in case of worst possible failure of estimates, LEDBAT will behave identically to TCP. This provides an extra safety net.

---

5. LEDBAT congestion control

TOC

Consider two parties, a sender and a receiver, with the sender having an unlimited source of data to send to the receiver and the receiver merely acknowledging the data. (In an actual protocol, it's more convenient to have bidirectional connections, but unidirectional abstraction suffices to describe the congestion control mechanism.)
Consider a protocol that uses packets of equal size and acknowledges each of them separately. (Variable-sized packets and delayed acknowledgements are possible and are being implemented, but complicate the exposition.)
Assume that each data packet contains a header field timestamp. The sender puts a timestamp from its clock into this field. Further assume that each acknowledgement packet contains a field delay. It is shown below how it is populated.
Slow start behavior is unchanged in LEDBAT. Note that rampup is faster in slow start than during congestion avoidance and so very conservative implementations MAY skip slow start altogether.
As far as congestion control is concerned, the receiver is then very simple and operates as follows, using a pseudocode:

```
on data_packet:
        remote_timestamp = data_packet.timestamp
        acknowledgement.delay = local_timestamp() - remote_timestamp
        # fill in other fields of acknowledgement
        acknowlegement.send()
```

The sender actually operates the congestion control algorithm and acts, in first approximation, as follows:

```
on acknowledgement:
        current_delay = acknowledgement.delay
        base_delay = min(base_delay, current_delay)
        queuing_delay = current_delay - base_delay
        off_target = TARGET - queuing_delay
        cwnd += GAIN * off_target / cwnd
```

The pseudocode above is a simplification and ignores noise filtering and base expiration. The more precise pseudocode that takes these factors into account is as follows and MUST be followed:

```
on acknowledgement:
        delay = acknowledgement.delay
        update_base_delay(delay)
        update_current_delay(delay)
        queuing_delay = current_delay() - base_delay()
        off_target = TARGET - queuing_delay
        cwnd += GAIN * off_target / cwnd

update_current_delay(delay)
        # Maintain a list of NOISE_FILTER last delays observed.
        forget the earliest of NOISE_FILTER current_delays
        add delay to the end of current_delays

current_delay()
        min(the NOISE_FILTER delays stored by update_current_delay)

update_base_delay(delay)
        # Maintain BASE_HISTORY min delays. Each represents a minute.
        if round_to_minute(now) != round_to_minute(last_rollover)
                last_rollover = now
                forget the earliest of base delays
                add delay to the end of base_delays
        else
                last of base_delays = min(last of base_delays, delay)

base_delay()
        min(the BASE_HISTORY min delays stored by update_base_delay)
```

TARGET parameter MUST be set to 25 milliseconds and GAIN MUST be set so that max rampup rate is the same as for TCP. BASE_HISTORY MUST be no less than 2 and SHOULD NOT be more than 10. NOISE_FILTER SHOULD be tuned so that it is at least 1 and no more than cwnd/2.

---

6. Security Considerations

TOC

An network on the path might choose to cause higher delay measurements than the real queuing delay so that LEDBAT backs off even when there's no congestion present. Shaping of traffic into an artificially narrow bottleneck can't be counteracted, but faking timestamp field can and SHOULD. A protocol using the LEDBAT congestion control SHOULD authenticate the timestamp and delay fields, preferably as part of authenticating most of the rest of the packet, with the exception of volatile header fields. The choice of the authentication mechanism that resists man-in-the-middle attacks is outside of scope of this document.

---

7. Normative References

TOC

[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

Author's Address

TOC

	Stanislav Shalunov
	BitTorrent Inc
	612 Howard St, Suite 400
	San Francisco, CA 94105
	USA
Email:	shalunov@bittorrent.com
URI:	http://shlang.com