LEDBAT WG
Internet-Draft
Intended status: Experimental
Expires: April 11, 2012

S. Shalunov
G. Hazel
BitTorrent Inc
J. Iyengar
Franklin and Marshall College
M. Kuehlewind
University of Stuttgart

October 9, 2011

Low Extra Delay Background Transport (LEDBAT)
draft-ietf-ledbat-congestion-08.txt

Abstract

LEDBAT is an experimental delay-based congestion control algorithm that attempts to utilize the available bandwidth on an end-to-end path while limiting the consequent increase in queueing delay on the path. LEDBAT uses changes in one-way delay measurements to limit congestion that the flow itself induces in the network. LEDBAT is designed for use by background bulk-transfer applications; it is designed to be no more aggressive than TCP congestion control and to yield in the presence of any competing flows when latency builds, thus limiting interference with the network performance of the competing flows.

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 http://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 11, 2012.

Table of Contents

1. Requirements notation

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].

2. Introduction

TCP congestion control [RFC5681] seeks to share bandwidth at a bottleneck link equitably among flows competing at the bottleneck, and it is the predominant congestion control mechanism used on the Internet. Not all applications seek an equitable share of network throughput, however "background" applications, such as software updates or file-sharing applications, seek to operate without interfering with the performance of more interactive and delay- and/or bandwidth-sensitive "foreground" applications and standard TCP may be too aggressive for use with such background applications.

LEDBAT is an experimental delay-based congestion control mechanism that reacts early to congestion in the network, thus enabling "background" applications to use the network while avoidoing interference with the network performance of competing flows. A LEDBAT sender uses one-way delay measurements to estimate the amount of queueing on the data path, controls the LEDBAT flow's congestion window based on this estimate, and minimizes interference with competing flows when latency builds by adding low extra queueing delay on the end-to-end path.

Delay-based congestion control protocols, such as TCP-Vegas [Bra94][Low02], are generally designed to achieve more, not less throughput than standard TCP, and often outperform TCP under particular network settings. In contrast, LEDBAT is designed to be no more aggressive than TCP; LEDBAT is a "scavenger" congestion control mechanism that seeks to utilize all available bandwidth and yields quickly when competing with standard TCP at a bottleneck link.

2.1. Design Goals

LEDBAT congestion control seeks to:

1. utilize end-to-end available bandwidth, and maintain low queueing delay when no other traffic is present,
2. add little to the queuing delay induced by concurrent flows,
3. quickly yield to flows using standard TCP congestion control that share the same bottleneck link,

2.2. Applicability

LEDBAT is a "scavenger" congestion control mechanism that is primarily motivated by background bulk-transfer applications, such as large file transfers (as with file-sharing applications) and software updates. It can be used with any application that seeks to minimize its impact on the network and on other interactive delay- and/or bandwidth-sensitive network applications. LEDBAT is expected to work well when the sender and/or receiver is connected via a residential access network.

LEDBAT can be used as part of a transport protocol or as part of an application, as long as the data transmission mechanisms are capable of carrying timestamps and acknowledging data frequently. LEDBAT can be used, with appropriate extensions where necessary, with TCP, SCTP, and DCCP, and with proprietary application protocols such as those built on top of UDP for P2P applications.

When used with an ECN-capable framing protocol, LEDBAT should react to an ECN mark as it would to a loss, as specified in [RFC3168].

LEDBAT is designed to reduce build-up of a standing queue by long-lived LEDBAT flows at a link with a tail-drop FIFO queue, so as to avoid persistently delaying other flows sharing the queue. If Active Queue Management (AQM) is configured to drop or ECN-mark packets before the LEDBAT starts reacting to persistent queue build-up, LEDBAT reverts to standard TCP behavior, rather than yield to other TCP flows. However, such an AQM is still desirable since it keeps queuing delay low, achieving an outcome that is in line with LEDBAT's

goals. Additionally, a LEDBAT transport that supports ECN enjoys the advantages that an ECN-capable TCP enjoys over an ECN-agnostic TCP; avoiding losses and possible retransmissions. Weighted Fair Queuing (WFQ), as employed by some home gateways [Schneider10], seeks to isolate and protect delay-sensitive flows from delays due to standing queues built up by concurrent long-lived flows. Consequently, while it prevents LEDBAT from yielding to other TCP flows, it again achieves an outcome that is in line with LEDBAT's goals [Schneider10].

Further study is required to fully understand the behaviour of LEDBAT with non-drop-tail, non-FIFO queues.

3. LEDBAT Congestion Control

3.1. Overview

A standard TCP sender increases its congestion window until a loss occurs [RFC5681] or an ECN mark is received [RFC3168], which, in the absence of any Active Queue Management (AQM) and link errors in the network, occurs only when the queue at the bottleneck link on the end-to-end path overflows. Since packet loss or marking at the bottleneck link is expected to be preceded by an increase in the queueing delay at the bottleneck link, LEDBAT congestion control uses this increase in queueing delay as an early signal of congestion, enabling it to respond to congestion earlier than standard TCP, and enabling it to yield bandwidth to a competing TCP flow.

LEDBAT employs one-way delay measurements to estimate queueing delay. When the estimated queueing delay is less than a pre-determined target, LEDBAT infers that the network is not yet congested, and increases its sending rate to utilize any spare capacity in the network. When the estimated queueing delay becomes greater than a pre-determined target, LEDBAT decreases its sending rate quickly as a response to potential congestion in the network.

3.2. Preliminaries

A LEDBAT sender uses a congestion window (cwnd) to gate the amount of data that the sender can send into the network in one roundtrip time (RTT). A sender MAY maintain its cwnd in bytes or in packets; this document uses cwnd in bytes. LEDBAT requires that each data segment carries a "timestamp" from the sender, based on which the receiver computes the one-way delay from the sender, and sends this computed value back to the sender.

In addition to the LEDBAT mechanism described below, we note that a slow start mechanism can be used as specified in [RFC5681]. Since slow start leads to faster increase in the window than that specified in LEDBAT, conservative congestion control implementations employing LEDBAT may skip slow start altogether and start with an initial window of INIT_CWND * MSS. (INIT_CWND is described later in Section 3.5.)

The term "MSS", or the sender's Maximum Segment Size, used in this document refers to the size of the largest segment that the sender can transmit. The value of MSS can be based on the path MTU discovery [RFC4821] algorithm and/or on other factors.

3.3. Receiver-Side Operation

A LEDBAT receiver operates as follows:


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

A receiver MAY send more than one delay sample in an acknowledgment. For instance, a receiver that delays acknowledgments, i.e., sends an acknowledgment less frequently than once per data packet, MAY send all the one-way delay samples that it gathers in one acknowledgment.

3.4. Sender-Side Operation

3.4.1. An Overview

As a first approximation, a LEDAT sender operates as shown below; the complete algorithm is specified later in Section 3.4.2. TARGET is the maximum queueing delay that LEDBAT itself can introduce in the network, and GAIN determines the rate at which the cwnd responds to changes in queueing delay; both constants are specified later. Since off_target can be positive or negative, the cwnd increases or decreases in proportion to off_target.

```
on initialization:
    base_delay = +INFINITY

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

The simplified mechanism above ignores multiple delay samples in an acknowledgment, noise filtering, base delay expiration, and sender idle times, which we now take into account in our complete sender algorithm below.

3.4.2. The Complete Sender Algorithm

update_current_delay() maintains a list of one-way delay measurements, of which a filtered value is used as an estimate of the current end-to-end delay. update_base_delay() maintains a list of one-way delay minima over a number of one-minute intervals, to measure and to track changes in the base delay of the end-to-end path.

This algorithm restricts cwnd growth after a period of inactivity,

where the cwnd is clamped down to a little more than flightsize using max_allowed_cwnd. To be TCP-friendly on data loss, LEDBAT halves its cwnd. The full sender-side algorithm is given below:

```
on initialization:
    # cwnd is the amount of data that is allowed to send in one RTT and
    # is defined in bytes.
    # CTO is the Congestion Timeout value.
```

```
    create current_delays list with CURRENT_FILTER elements
    create base_delays list with BASE_HISTORY number of elements
    initialize elements in current_delays and base_delays to +INFINITY
    last_rollover = -INFINITY # More than a minute in the past
    flightsize = 0
    cwnd = INIT_CWND * MSS
    CTO = 1 second

on acknowledgment:
    # flightsize is the amount of data outstanding before this ack
    #    was received and is updated later;
    # bytes_newly_acked is the number of bytes that this ack
    #    newly acknowledges, and it MAY be set to MSS.

    for each delay sample in the acknowledgment:
        delay = acknowledgement.delay
        update_base_delay(delay)
        update_current_delay(delay)
    queuing_delay = FILTER(current_delays) - MIN(base_delays)
    off_target = (TARGET - queuing_delay) / TARGET
    cwnd += GAIN * off_target * bytes_newly_acked * MSS / cwnd
    max_allowed_cwnd = flightsize + ALLOWED_INCREASE * MSS
    cwnd = min(cwnd, max_allowed_cwnd)
    cwnd = max(cwnd, MIN_CWND * MSS)
    flightsize = flightsize - bytes_newly_acked
    update_CTO()

on data loss:
    # atmost once per RTT
    cwnd = min (cwnd, max (cwnd/2, MIN_CWND * MSS))
    if data lost is not to be retransmitted:
        flightsize = flightsize - bytes_not_to_be_retransmitted

if no acks are received within a CTO:
    # extreme congestion, or significant RTT change.
    # set cwnd to 1MSS and backoff the congestion timer.
    cwnd = 1 * MSS
    CTO = 2 * CTO
```


update_CTO()

```
    # implements an RTT estimation mechanism using data
    # transmission times and ack reception times,
    # which is used to implement a congestion timeout (CTO).
    # If implementing in TCP, sender SHOULD use
    # mechanisms described in RFC 6298 <xref target='RFC3390'/>,
    # and the CTO is the same as the RTO.

update_current_delay(delay)
    # Maintain a list of CURRENT_FILTER last delays observed.
    delete first item in current_delays list
    append delay to current_delays list

update_base_delay(delay)
    # Maintain BASE_HISTORY delay-minima.
    # Each minimum is measured over a period of a minute.
    # 'now' is the current system time
    if round_to_minute(now) != round_to_minute(last_rollover)
        last_rollover = now
        delete first item in base_delays list
        append delay to base_delays list
    else
        base_delays.tail = MIN(base_delays.tail, delay)
```

The LEDBAT sender ensures that any outliers in the current_delay samples are eliminated by implementing FILTER() to extract the actual delay estimate from the current_delay samples. An example of such a filter is the Exponentially-Weighted Moving Average (EWMA) function.

To implement an approximate minimum over the past few minutes, a LEDBAT sender stores BASE_HISTORY separate minima---one each for the last BASE_HISTORY-1 minutes, and one for the running current minute. At the end of the current minute, the window moves---the earliest minimum is dropped and the latest minimum is added. If the connection is idle for a given minute, no data is available for the one-way delay and, therefore, a value of +INFINITY has to be stored in the list. If the connection has been idle for BASE_HISTORY minutes, all minima in the list are thus set to +INFINITY and measurement begins anew. LEDBAT thus requires that during idle periods, an implementation must maintain the base delay list.

LEDBAT uses a congestion timeout (CTO) to avoid transmitting data during periods of heavy congestion, and to avoid congestion collapse. A CTO is used to detect heavy congestion indicated by loss of all outstanding data or acknowledgments, resulting in reduction of the cwnd to 1 MSS and an exponential backoff of the CTO interval. This

backoff of the CTO value avoids sending more data into an overloaded queue, and also allows the sender to cope with sudden changes in the RTT of the path. The function of a CTO is similar to that of an retransmission timeout (RTO) in TCP [RFC3390], but since LEDBAT separates reliability from congestion control, a retransmission need not be triggered by a CTO. LEDBAT, however does not preclude a CTO from triggering retransmissions, as could be the case if LEDBAT congestion control were to be used with TCP framing and reliability.

The CTO is a gating mechanism that ensures exponential backoff of sending rate under heavy congestion, and it may be implemented with or without a timer. An implementation choosing to avoid timers may consider using a "next-time-to-send" variable, set based on the CTO, to control the earliest time a sender may transmit without receiving any acks.

A maximum value MAY be placed on the CTO, and if placed, it MUST be 60 seconds or more.

We note that LEDBAT assumes random fluctuations in inter-packet transmission times. That will help to measure the correct base delay beacuse the the bottleneck runs empty from time to time; see section Section 5.3 for a discussion.

3.5. Parameter Values

TARGET MUST be 100 milliseconds or less, and this choice of value is explained further in Section 4.3. Note that using the same TARGET value across LEDBAT flows enables equitable sharing of the bottleneck bandwidth---flows with a higher TARGET may get a larger share of the bottleneck bandwidth. It is possible to consider the use of different TARGET values for implementing a relative priority between two competing LEDBAT flows by setting a higher TARGET value for the higher-priority flow.

ALLOWED_INCREASE SHOULD be 1, and it MUST be greater than 0. An ALLOWED_INCREASE of 0 results in no cwnd growth at all, and an ALLOWED_INCREASE of 1 allows and limits the cwnd increase based on flightsize in the previous RTT. An ALLOWED_INCREASE greater than 1 MAY be used when interactions between LEDBAT and the framing protocol provide a clear reason for doing so.

GAIN MUST be set to 1 or less. A GAIN of 1 limits the maximum cwnd ramp-up to the same rate as TCP Reno in Congestion Avoidance. While this document specifies the use of the same GAIN for both cwnd increase (when off_target is greater than zero) and decrease (when

off_target is less than zero), implementations MAY use a higher GAIN for cwnd decrease than for the increase; our justification follows.

When a competing non-LEDBAT flow increases its sending rate, the LEDBAT sender may only measure a small amount of additional delay and decrease the sending rate slowly. To ensure no impact on a competing non-LEDBAT flow, the LEDBAT flow should decrease its sending rate at least as quickly as the competing flow increases its sending rate. A higher decrease GAIN MAY be used to allow the LEDBAT flow to decrease its sending rate faster than the competing flow's increase rate.

The size of the base_delays list, BASE_HISTORY, SHOULD be 10. If the actual base delay decreases, due to a route change for instance, a LEDBAT sender adapts immediately, irrespective of the value of BASE_HISTORY. If the actual base delay increases however, a LEDBAT sender will take BASE_HISTORY minutes to adapt and may wrongly infer a little more extra delay than intended (TARGET) in the meanwhile. A value for BASE_HISTORY is thus a tradeoff: a higher value may yield a more accurate measurement when the base delay is unchanging, and a lower value results in a quicker response to actual increase in base delay.

A LEDBAT sender uses the current_delays list to maintain only delay measurements made within a RTT amount of time in the past, seeking to eliminate noise spikes in its measurement of the current one-way delay through the network. The size of this list, CURRENT_FILTER, may be variable, and depends on the number of successful measurements made within a RTT amount of time in the past. The sender should seek to gather enough delay samples in each RTT so as to have statistical confidence in the measurements. While the number of delay samples required for such confidence will vary depending on network conditions, we recommend that the sender SHOULD use at least 4 samples in each RTT, unless the number of samples is lower due to a small congestion window. Thus, subject to congestion window constraints, the number of delay samples in each RTT SHOULD be at least 4, and thus CURRENT_FILTER SHOULD be at least 4. CURRENT_FILTER MUST be limited such that samples in the list are not older than an RTT in the past.

INIT_CWND SHOULD be 4, and MIN_CWND SHOULD be 2. An INIT_CWND of 4 should help seed FILTER() at the sender when there are no samples at the beginning of a flow, and a MIN_CWND of 2 allows FILTER() to use

more than a single instantaneous delay estimate while not being too aggressive. Slight deviations may be warranted, for example, when these values of MIN_CWND and INIT_CWND interact poorly with the framing protocol.

4. Understanding LEDBAT Mechanisms

This section describes the delay estimation and window management

mechanisms used in LEDBAT.

4.1. Delay Estimation

LEDBAT estimates congestion in the direction of the data flow, and to avoid measuring additional delay from e.g. queue build-up on the reverse path (or ack path) or reordering, LEDBAT uses one-way delay estimates. LEDBAT assumes measurements are done with data packets, thus avoiding the need for separate measurement packets and avoiding the pitfall of measurement packets being treated differently from the data packets in the network.

End-to-end delay can be decomposed into transmission (or serialization) delay, propagation (or speed-of-light) delay, queueing delay, and processing delay. On any given path, barring some noise, all delay components except for queueing delay are constant. To observe an increase in the queueing delay in the network, a LEDBAT sender separates the queueing delay component from the rest of the end-to-end delay, as described below.

4.1.1. Estimating Base Delay

Since queuing delay is always additive to the end-to-end delay, LEDBAT estimates the sum of the constant delay components, which we call "base delay", to be the minimum delay observed on the end-to-end path.

To respond to true changes in the base delay, as can be caused by a route change, LEDBAT uses only recent measurements in estimating the base delay. The duration of the observation window itself is a tradeoff between robustness of measurement and responsiveness to change---a larger observation window increases the chances that the

true base delay will be detected (as long as the true base delay is unchanged), whereas a smaller observation window results in faster response to true changes in the base delay.

4.1.2. Estimating Queueing Delay

Given that the base delay is constant, the queueing delay is represented by the variable component of the measured end-to-end delay. LEDBAT measures queueing delay as simply the difference between an end-to-end delay measurement and the current estimate of base delay. The queueing delay should be filtered (depending on the usage scenario) to eliminate noise in the delay estimation, such as due to spikes in processing delay at a node on the path.

4.2. Managing the Congestion Window

4.2.1. Window Increase: Probing For More Bandwidth

A LEDBAT sender increases its congestion window if the queuing delay is smaller than a target value, proportionally to the relative difference between the current queueing delay and the delay target. To be friendly to competing TCP flows, we set this highest rate of window growth to be the same as TCP's. In other words, a LEDBAT flow thus never ramps up faster than a competing TCP flow over the same path. As closer the extra delay gets to the TARGET value, as slower LEDBAT will increase the window.

4.2.2. Window Decrease: Responding To Congestion

When the sender's queueing delay estimate is higher than the target, the LEDBAT flow's rate should be reduced. LEDBAT uses a simple linear controller to detemine sending rate as a function of the delay estimate, where the response is proportional to the difference between the current queueing delay estimate and the target. This allows to decrease the window only slightly while probing and leads to a quite stable state with high link utilization. In limited experiments with Bittorrent nodes, this controller seems to work well.

Unlike TCP-like loss-based congestion control, LEDBAT seeks to avoid losses and so a LEDBAT sender is not expected to normally rely on losses to determine the sending rate. However, when data loss does occur, LEDBAT must respond as standard TCP does; even if the queueing delay estimates indicate otherwise, a loss is assumed to be a strong indication of congestion. Thus, to deal with severe congestion when packets are dropped in the network, and to provide a fallback against incorrect queuing delay estimates, a LEDBAT sender halves its congestion window when a loss event is detected. As with TCP New-Reno, LEDBAT reduces its cwnd by half at most once per RTT.

4.3. Choosing The Queuing Delay Target

The queueing delay target is a tradeoff. A target that is too low might result in under-utilization of the bottleneck link, because of the noise in the delay measurement e.g in a mobile scenario, and may also be more sensitive to error in the measured delay. The International Telecommunication Union's (ITU's) Recommendation G.114 defines a delay of 150 ms to be acceptable for most user voice applications. Thus the extra delay induced by LEDBAT must be below 150 ms to reduce impact on delay-sentive applications. If the TARGET value is larger than the maximum delay the queue can induce, LEBAT will fallback to the same behavior than standard TCP (see section

Section 5.2).

Our recommendation of 100 ms or less as the target is based on these considerations. Anecdotal evidence indicates that this value works well: LEDBAT has been implemented and successfully deployed with a target value of 100 ms in two Bittorrent implementations---BitTorrent DNA as the exclusive congestion control mechanism and in uTorrent as an experimental mechanism.

5. Discussion

5.1. Framing and Ack Frequency Considerations

While the actual framing and wire format of the protocols using LEDBAT are outside the scope of this document, we briefly consider the data framing and ack frequency needs of LEDBAT mechanisms.

To compute the data path's one-way delay, our discussion of LEDBAT assumes a framing that allows the sender to timestamp packets and for the receiver to convey the measured one-way delay back to the sender in ack packets. LEDBAT does not require this particular method, but it does require unambiguous delay estimates using data and ack packets.

A LEDBAT receiver may send an ack as frequently as one for every data packet received or less frequently; LEDBAT does require that the receiver MUST transmit at least one ack in every RTT.

5.2. Competing With TCP Flows

LEDBAT is designed to respond to congestion indications earlier than loss-based TCP. A LEDBAT flow is more aggressive when the queueing delay estimate is lower; since the queueing delay estimate is non-negative, LEDBAT is most aggressive when its queuing delay estimate is zero. In this case, LEDBAT ramps up its congestion window at the same rate as TCP does. LEDBAT reduces its rate earlier than TCP does, always halving the congestion window on loss. Thus, in the worst case where the delay estimates are completely and consistently off, a LEDBAT flow falls back to TCP mechanisms as it will be at most as aggressive as a TCP flow and halves on loss.

5.3. Fairness Among LEDBAT Flows

The primary design goals of LEDBAT are focussed on the aggregate behavior of LEDBAT flows when they compete with standard TCP. Since LEDBAT is designed for background traffic, we consider link utilization to be more important than fairness amongst LEDBAT flows.

Nevertheless, we now consider fairness issues that might arise amongst competing LEDBAT flows.

LEDBAT as described so far lacks a mechanism specifically designed to equalize utilization amongst LEDBAT flows. Anecdotally observed behavior of existing implementations indicates that a rough equalization does occur since in most enviroments some amount of randomness in the inter-packet transmission times exist, as explained further below.

Delay-based congestion control systems suffer from the possibility of

late-comers incorrectly measuring and using a higher base-delay than an active flow that started earlier. Suppose a LEDBAT flow is the only flow on the bottleneck, which the flow saturates, steadily maintaining the queueing delay at a target delay. When a new LEDBAT flow arrives, it might incorrectly measure the current end-to-end delay, including the queueing delay being maintained by the first LEDBAT flow, as its base delay, and the incoming flow might now effectively seek to build on top of the existing, already maximal queueing delay. As the second flow builds up, the first flow sees the true queueing delay and backs off, while the late-comer keeps building up, using up the entire link's capacity; this advantage is called the "late-comer's advantage".

In the worse case, if the first flow yields at the same rate as the new flow increases its sending rate, the new flow will see constant end-to-end delay, which it assumes is the base delay, until the first flow backs off completely. As a result, by the time the second flow stops increasing its cwnd, it would have added twice the target queueing delay to the network.

This advantage can be reduced if the the first flow yields quickly enough to empty the bottleneck queue faster than the incoming flow increases its occupancy in the queue; as a result, the late-comer might measure a delay closer to the base delay. While such a reduction might be achieved through a multiplicative decrease of the congestion window, this might cause stronger fluctuations in flow throughput during steady state. Thus we do not recommend a multiplicative decrease scheme.

We note that in certain use-case scenarios, it is possible for a later LEDBAT flow to gain an unfair advantage over an existing one [Car10]. In practice, this concern may be alleviated by the burstiness of network traffic: all that's needed to measure the base delay is one small gap in transmission schedules between the LEDBAT flows. These gaps can occur for a number of reasons such as latency introduced due to application sending patterns, OS scheduling at the sender, processing delay at the sender or any network node, and link

contention. When such a gap occurs in the first sender's transmission while the late-comer is starting, base delay is immediately correctly measured. With a small number of LEDBAT flows, system noise may sufficiently regulate the late-comer's advantage.

6. IANA Considerations

There are no IANA considerations for this document.

7. Security Considerations

A 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. While shaping of traffic into an artificially narrow bottleneck by increasing the queueing delay cannot be trivially counteracted, a protocol using LEDBAT should seek to minimize the risk of such an attack by authenticating the timestamp and delay fields in the packets.

LEDBAT is not known to introduce any new concerns with privacy, integrity, or other security issues for flows that use it. It should be compatible with use of IPsec and TLS/DTLS.

8. Acknowledgements

We thank folks in the LEDBAT working group for their comments and feedback. Special thanks to Murari Sridharan and Rolf Winter for their patient and untiring shepherding.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU

Discovery", RFC 4821, March 2007.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

9.2. Informative References

[Bra94] Brakmo, L., O'Malley, S., and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", Proceedings of SIGCOMM '94, pages 24-35, August 1994.

[Car10] Carofiglio, G., Muscariello, L., Rossi, D., Testa, C., and S. Valenti, "Rethinking Low Extra Delay Background Transport Protocols", arXiv:1010.5623v1, September 2010.

[Low02] Low, S., Peterson, L., and L. Wang, "Understanding TCP Vegas: A Duality Model", JACM 49 (2), March 2002.

Appendix A. Timestamp errors

One-way delay measurement needs to deal with timestamp errors. We'll use the same locally linear clock model and the same terminology as Network Time Protocol (NTP). This model is valid for any differentiable clocks. NTP uses the term "offset" to refer to difference from true time and "skew" to refer to difference of clock rate from the true rate. 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.

A.1. Clock offset

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.

A.2. Clock skew

The 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 small. 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 base delay updates mostly takes care of clock skew unless the skew is unusually high or extreme values have been chosen for TARGET and BASE_HISTORY so that the clock skew in BASE_DELAY minutes is larger than the TARGET.

Clock skew can be in two directions: either the sender's clock is faster than the receiver's, or vice versa.

If the sender's clock is faster the one-way delay measurement will get more and more reduced by the clock drift over time. Whenever there is no additional delay the base delay will be updated by a smaller one-way delay value and the skew is compensated. If a competing flow introduces additional queueing delay LEDBAT will anyway get out of the way quickly and an overestimated one-way delay will just speed-up the back-off.

When the receiver clock runs faster, the raw delay estimate will drift up with time. This can suppress the throughput unnecessarily. In this case a skew correction mechanim can be benefital. Further condersiderations based on a deployed implementation and LEDBAT specific preconditions are given in the next section.

A.3. Clock skew correction mechanism

The following paragraph describes the deployed clock skew correction mechanism in the BitTorrent implementation for documentation purpose.

In the BitTorrent implemtation the receiver sends back the raw (sending and receiving) timestamps. Based on this imfomation and the system time at feedback receiption the sender can estimated the one-way delay in both directions. Thus the sender can run the same base delay calculation algorithm it runs for itself for the receiver as well. If the sender can detect the receiver reducing its base delay, it can infer that this is due to clock drift. The sender can be

compensated for by increasing the it's base delay by the same amount. To apply this mechanism symmetrical framing is need (i.e., same information about delays transmitted in both way).

The following considerations can be used for an alternative implementation as a reference:

- Skew correction with faster virtual clock:
  Since having a faster clock on the sender will continuousely update the base delay, a faster virtual clock for sender timestamping can be applied. This virual clock can be computed from the default machine clock through a linear transformation. E.g. with a 500 PPM speed-up the sender's clock is very likely to be faster than any receiver's clock and thus LEDBAT will benefit from the implicit correction when updating the base delay.

- Skew correction with estimating drift:
  With LEDBAT the history of base delay minima is already kept for each minute. This can provide a base to compute the clock skew difference between the two hosts. The slope of a linear function fitted to the set of minima base delays gives an estimate of the clock skew. This estimation can be used to correct the clocks. If the other endpoint is doing the same, the clock should be corrected by half of the estimated skew amount.

- Byzantine skew correction:
  When it is known that each host maintains long-lived connections to a number of different other hosts, a byzantine scheme can be used to estimate the skew with respect to the true time. Namely, calculate the skew difference for each of the peer hosts as described with the previous approach, then take the median of the skew differences. While this scheme is not universally applicable, it combines well with other schemes, since it is essentially a clock training mechanism. The scheme also acts the fastest, since the state is preserved between connections.

Authors' Addresses

Stanislav Shalunov

BitTorrent Inc
612 Howard St, Suite 400
San Francisco, CA 94105
USA

Email: shalunov@bittorrent.com
URI: http://shlang.com

Greg Hazel
BitTorrent Inc
612 Howard St, Suite 400
San Francisco, CA 94105
USA

Email: greg@bittorrent.com

Janardhan Iyengar
Franklin and Marshall College
415 Harrisburg Ave.
Lancaster, PA 17603
USA

Email: jiyengar@fandm.edu

Mirja Kuehlewind
University of Stuttgart
Stuttgart
DE

Email: mirja.kuehlewind@ikr.uni-stuttgart.de