Internet Engineering Task Force (IETF)
Request for Comments: 6817
Category: Experimental
ISSN: 2070-1721

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

December 2012

Low Extra Delay Background Transport (LEDBAT)

Abstract

Low Extra Delay Background Transport (LEDBAT) is an experimental delay-based congestion control algorithm that seeks to utilize the available bandwidth on an end-to-end path while limiting the consequent increase in queueing delay on that 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 to be no more aggressive than standard TCP congestion control (as specified in RFC 5681) and to yield in the presence of competing flows, thus limiting interference with the network performance of competing flows.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. 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). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6817.

Table of Contents

1. 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. However, not all applications seek an equitable share of network throughput. "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. Standard TCP congestion control, as specified in [RFC5681], may be too aggressive for use with such background applications.

Low Extra Delay Background Transport (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 avoiding 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 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. For further discussion on Lower-than-Best-Effort transport protocols see [RFC6297]. In contrast, LEDBAT is designed to be no more aggressive than TCP [RFC5681]; 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.

In the rest of this document, we refer to congestion control specified in [RFC5681] as "standard TCP".

1.1. 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].

1.2. Design Goals

LEDBAT congestion control seeks to achieve the following goals:

1. to utilize end-to-end available bandwidth and to maintain low queueing delay when no other traffic is present,

2. to add limited queuing delay to that induced by concurrent flows, and

3. to yield quickly to standard TCP flows that share the same bottleneck link.

1.3. Applicability

LEDBAT is a "scavenger" congestion control mechanism that is motivated primarily 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 TCP, Stream Control Transmission Protocol (SCTP), and Datagram Congestion Control Protocol (DCCP), with appropriate extensions where necessary; and it can be used with proprietary application protocols, such as those built on top of UDP for peer-to-peer (P2P) applications.

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

LEDBAT is designed to reduce buildup 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 flow starts reacting to persistent queue buildup, LEDBAT reverts to standard TCP behavior rather than yielding 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, 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 [Sch10].

2. LEDBAT Congestion Control

2.1. 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 link errors in the network, occurs only when the queue at the bottleneck link on the end-to-end path overflows or an AQM is applied. 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 predetermined 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 the predetermined target, LEDBAT decreases its sending rate as a response to potential congestion in the network.

2.2. 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 round-trip 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 2.5.)

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.

2.3. Receiver-Side Operation

A LEDBAT receiver calculates the one-way delay from the sender to the receiver based on its own system time and timestamps in the received data packets. The receiver then feeds the computed one-way delay back to the sender in the next acknowledgement. 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
    acknowledgement.send()
```

A receiver may choose to delay sending an ACK and may combine acknowledgements for more than one data packet into a single ACK packet, as with delayed ACKs in standard TCP, for example. In such cases, the receiver MAY bundle all the delay samples into one ACK packet and MUST transmit the samples in the order generated. When multiple delay samples are bundled within a single ACK, the sender applies these bundled delay samples at once during its cwnd adjustment (discussed in the next section). Since the sender's adjustment may be sensitive to the order in which the delay samples are applied, the computed delay samples should be available to the sender in the order they were generated at the receiver.

2.4. Sender-Side Operation

2.4.1. An Overview

As a first approximation, a LEDBAT sender operates as shown below; the complete algorithm is specified later in Section 2.4.2. TARGET is the maximum queueing delay that LEDBAT itself may introduce in the network, and GAIN determines the rate at which the cwnd responds to changes in queueing delay; both constants are specified later. off_target is a normalized value representing the difference between the measured current queueing delay and the predetermined TARGET delay. off_target can be positive or negative; consequently, 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 acknowledgement, noise filtering, base delay expiration, and sender idle times, which we now take into account in our complete sender algorithm below.

2.4.2. 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. Both of these lists are maintained per LEDBAT flow.

We note this algorithm assumes that slight random fluctuations exist in inter-packet arrival times at the bottleneck queue, to allow a LEDBAT sender to correctly measure the base delay. See Section 4.4 for a more complete discussion.

The full sender-side algorithm is given below:

```
on initialization:
    # cwnd is the amount of data that is allowed to be
    # outstanding in an 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 base_delays to +INFINITY
    initialize elements in current_delays according to FILTER()
    last_rollover = -INFINITY # More than a minute in the past
    flightsize = 0
    cwnd = INIT_CWND * MSS
    CTO = 1 second
```

```
   on acknowledgement:
       # 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 acknowledgement:
           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:
       # at most 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 LEDBAT in TCP, sender SHOULD use
    # mechanisms described in RFC 6298 [RFC6298],
    # and the CTO would be the same as the retransmission timeout (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 seeks to extract the actual delay estimate from the current_delay samples by implementing FILTER() to eliminate any outliers. Different types of filters MAY be used for FILTER() -- a NULL filter, that does not filter at all, is a reasonable candidate as well, since LEDBAT's use of a linear controller for cwnd increase and decrease may allow it to recover quickly from errors induced by bad samples. Another example of a filter is the exponentially weighted moving average (EWMA) function, with weights that enable agile tracking of changing network delay. A simple MIN filter applied over a small window (much smaller than BASE_HISTORY) may also provide robustness to large delay peaks, as may occur with delayed ACKs in TCP. Care should be taken that the filter used, while providing robustness to noise, remains sensitive to persistent congestion signals.

We note that when multiple delay samples are bundled within a single ACK, the sender's resulting cwnd may be slightly different than when the samples are sent individually in separate ACKs. The cwnd is adjusted based on the total number of bytes ACKed and the final filtered value of queueing_delay, irrespective of the number of delay samples in an ACK.

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 restricts cwnd growth after a period of inactivity. When the sender is application-limited, the sender's cwnd is clamped down using max_allowed_cwnd to a little more than flightsize. To be TCP-friendly, LEDBAT halves its cwnd on data loss.

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 acknowledgements, 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 it 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 [RFC6298], 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 at least 60 seconds.

2.5. Parameter Values

TARGET MUST be 100 milliseconds or less, and this choice of value is explained further in Section 3.3. Note that using the same TARGET value across LEDBAT flows enables equitable sharing of the bottleneck bandwidth. A flow with a higher TARGET value than other competing LEDBAT flows 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 trade-off: 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 an 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 it depends on the FILTER() function as well as the number of successful measurements made within an 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, the sender SHOULD use at least 4 delay samples in each RTT, unless the number of samples is lower due to a small congestion window. The value of CURRENT_FILTER will depend on the filter being employed, but CURRENT_FILTER MUST be limited such that samples in the list are not older than an RTT in the past.

INIT_CWND and MIN_CWND SHOULD both be 2. An INIT_CWND of 2 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 INIT_CWND and MIN_CWND interact poorly with the framing protocol. However, INIT_CWND and MIN_CWND MUST be no larger than the corresponding values specified for TCP [RFC5681].

3. Understanding LEDBAT Mechanisms

This section describes the delay estimation and window management mechanisms used in LEDBAT.

3.1. Delay Estimation

LEDBAT estimates congestion in the direction of the data flow, and to avoid measuring additional delay from, e.g., queue buildup on the reverse path (or ACK path) or reordering, LEDBAT uses one-way delay estimates. LEDBAT assumes that 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.

3.1.1. 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 trade-off 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.

3.1.2. Estimating Queueing Delay

Assuming that the base delay is constant (in the absence of any route changes), 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.

3.2. Managing the Congestion Window

LEDBAT uses a simple linear controller to determine the sending rate as a function of the delay estimate, where the response of the sender is proportional to the difference between the current queueing delay estimate and the target.

3.2.1. Window Increase: Probing for More Bandwidth

When the queuing delay is smaller than a delay target value, as specified by the TARGET parameter in this document, a LEDBAT sender will increase its congestion window proportionally to the relative difference between the current queueing delay and the delay target. As the current queuing delay gets closer to TARGET, LEDBAT's window growth gets slower. To compete fairly with concurrent TCP flows, we set the highest rate of LEDBAT's window growth (when the current queueing delay estimate is zero) to be the same as TCP's (increase of one packet per RTT). In other words, a LEDBAT flow never ramps up faster than a competing TCP flow over the same path. The TARGET value specifies the maximum extra queuing delay that LEDBAT will induce. If the current queuing delay equals the TARGET value, LEDBAT tries to maintain this extra delay.

3.2.2. Window Decrease: Responding to Congestion

When a sender's queueing delay estimate is higher than the target, the LEDBAT flow's rate should be reduced. LEDBAT's linear controller allows the sender to decrease the window proportional to the difference between the target and the current queueing delay.

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.

3.3. Choosing the Queuing Delay Target

The International Telecommunication Union's (ITU's) Recommendation G.114 defines a one-way delay of 150 ms to be acceptable for most user voice applications [g114]. Thus, the delay induced by LEDBAT must be well below 150 ms to limit its impact on concurrent delay-sensitive traffic sharing the same bottleneck queue. A target that is too low, on the other hand, increases the sensitivity of the sender's algorithm to noise in the one-way delays and in the delay measurement process, and may lead to reduced throughput for the LEDBAT flow and to under-utilization of the bottleneck link.

Our recommendation of 100 ms or less as the target is a trade-off between 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: as the exclusive congestion control mechanism in BitTorrent Delivery Network Accelerator (DNA), and as an experimental mechanism in uTorrent [uTorrent].

4. Discussion

4.1. 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.

4.2. Competing with TCP Flows

LEDBAT is designed to respond to congestion indications earlier than loss-based standard TCP [RFC5681]. A LEDBAT flow gets more aggressive as the queueing delay estimate gets lower; since the queueing delay estimate is non-negative, LEDBAT is most aggressive when the queueing delay estimate is zero. In this case, LEDBAT ramps up its congestion window at the same rate as standard TCP [RFC5681]. LEDBAT may reduce its rate earlier than standard TCP and always

halves its congestion window on loss. Thus, in the worst case, where the delay estimates are completely and consistently off, a LEDBAT flow falls back to standard TCP behavior, and is no more aggressive than standard TCP [RFC5681].

4.3. Competing with Non-TCP Flows

While LEDBAT yields to all high-load flows, both TCP and non-TCP, LEDBAT may not yield to low-load and latency-sensitive traffic that do not induce a measurable delay at the bottleneck queue, such as Voice over IP (VoIP) traffic. While such flows will experience additional delay due to any concurrent LEDBAT flows, the TARGET delay sets a limit to the total amount of additional delay that all the concurrent LEDBAT flows will jointly induce. If the TARGET delay is higher than what the bottleneck queue can sustain, the LEDBAT flows should experience loss and will fall back to standard loss-based TCP behavior. Thus, in the worst case, LEDBAT will add no more latency than standard TCP when competing with non-TCP flows. In the common case however, we expect LEDBAT flows to add TARGET amount of delay, which ought to be within the delay tolerance for most latency-sensitive applications, including VoIP applications.

4.4. Fairness among LEDBAT Flows

The primary design goals of LEDBAT are focused 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 environments some amount of randomness in the inter-packet transmission times exists, as explained further below.

Delay-based congestion control systems suffer from the possibility of latecomers incorrectly measuring and using a higher base-delay than an active flow that started earlier. Consider that a bottleneck is saturated by a single LEDBAT flow, and the flow therefore maintains the bottleneck queue at TARGET delay. When a new LEDBAT flow arrives at the bottleneck, it might incorrectly include the steady queueing delay in its measurement of the base delay on the path. The new flow has an inflated estimate of the base delay, and may now effectively build on top of the existing, already maximal, queueing delay. As the latecomer flow builds up, the old flow sees the true queueing

delay and backs off, while the latecomer keeps building up, using up the entire link's capacity, and effectively shutting the old flow out. This advantage is called the "latecomer's advantage".

In the worst 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 first flow yields and empties the bottleneck queue faster than the incoming flow increases its occupancy in the queue. In such a case, the latecomer might measure correctly a delay that is closer to the base delay. While such a reduction might be achieved through a multiplicative decrease of the congestion window, this may cause strong fluctuations in flow throughput during the flow's 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 ought to 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 latecomer is starting, base delay is immediately correctly measured. With a small number of LEDBAT flows, system noise may sufficiently regulate the latecomer's advantage.

5. Open Areas for Experimentation

We now outline some areas that need experimentation in the Internet and under different network scenarios. These experiments should help the community understand LEDBAT's dynamics and should help towards further standardization of LEDBAT and LEDBAT-related documents.

5.1. Network Effects and Monitoring

Further study is required to fully understand the behavior and convergence properties of LEDBAT in networks with non-tail-drop, non-FIFO queues, in networks with frequent route changes, and in networks with network-level load balancing. These studies should have two

broad goals: (i) to understand the effects of different network mechanisms on LEDBAT, and (ii) to understand the impact of LEDBAT on the network.

Network mechanisms and dynamics can influence LEDBAT flows in unintended ways. For instance, frequent route changes that result in increasing base delays may, in the worst case, throttle a LEDBAT flow's throughput significantly. The influence of different network traffic management mechanisms on LEDBAT throughput should be studied.

An increasing number of LEDBAT flows in the network will likely result in operator-visible network effects as well, and these should thus be studied. For instance, as long as the bottleneck queue in a network is larger than TARGET (in terms of delay), we expect that both the average queueing delay and loss rate in the network should reduce as LEDBAT traffic increasingly dominates the traffic mix in the network. Note that for bottleneck queues that are smaller than TARGET, LEDBAT will appear to behave very similar to standard TCP and its flow-level behavior may not be distinguishable from that of standard TCP.

We note that a network operator may be able to verify the operation of a LEDBAT flow by monitoring per-flow behavior and queues in the network -- when the queueing delay at a bottleneck queue is above TARGET as specified in this document, LEDBAT flows should be expected to back off and reduce their sending rate.

5.2. Parameter Values

The throughput and response of LEDBAT to the proposed parameter values of TARGET, decrease-GAIN, BASE_HISTORY, INIT_CWND, and MIN_CWND should be evaluated with different types of competing traffic in different network settings, including with different AQM schemes at the bottleneck queue. TARGET controls LEDBAT's added latency, while decrease-GAIN controls LEDBAT's response to competing traffic. Since LEDBAT is intended to be minimally intrusive to competing traffic, the impact of TARGET and decrease-GAIN on delay-sensitive traffic should be studied. TARGET also impacts the growth rate of the congestion window when off_target is smaller than 1. This impact of TARGET on the rate of cwnd growth should be studied. The amount of history maintained by the base delay estimator, BASE_HISTORY, influences the responsiveness of LEDBAT to changing network conditions. LEDBAT's responsiveness and throughput should be evaluated in the wide area and under conditions where abrupt changes in base delay might occur, such as with route changes and with cellular handovers. The impact and efficacy of these parameters should be carefully studied with tests over the Internet.

5.3. Filters

LEDBAT's effectiveness depends on a sender's ability to accurately estimate end-to-end queueing delay from delay samples. Consequently, the filtering algorithm used for this estimation, FILTER(), is an important candidate for experiments. This document suggests the use of NULL, EWMA, and MIN filters for estimating the current delay; the efficacy of these and other possible filters for this estimate should be investigated. FILTER() may also impact cwnd dynamics when delay samples are bundled in ACKs, since cwnd adaption is done once per ACK irrespective of the number of delay samples in the ACK. This impact should be studied when the different filters are considered.

5.4. Framing

This document defines only a congestion control algorithm and assumes that framing mechanisms for exchanging delay information exist within the protocol in which LEDBAT is being implemented. If implemented in a new protocol, both the sender and receiver may be LEDBAT-aware, but if implemented in an existing protocol that is capable of providing one-way delay information, LEDBAT may be implemented as a sender-side-only modification. In either case, the parent protocol may interact with LEDBAT's algorithms; for instance, the rate of ACK feedback to the data sender may be dictated by other protocol parameters, but will interact with the LEDBAT flow's dynamics. Careful experimentation is necessary to understand and integrate LEDBAT into both new and existing protocols.

6. Security Considerations

LEDBAT's aggressiveness is contingent on the delay estimates and on the TARGET delay value. If these parameter values at the sender are compromised such that delay estimates are artificially set to zero and the TARGET delay value is set to +INFINITY, the LEDBAT algorithm deteriorates to TCP-like behavior. Thus, while LEDBAT is sensitive to these parameters, the algorithm is fundamentally limited in the worst case to be as aggressive as standard TCP.

A man in the middle may be able to change queueing delay on a network path, and/or modify the timestamps transmitted by a LEDBAT sender and/or modify the delays reported by a LEDBAT receiver, thus causing a LEDBAT flow to back off even when there's no congestion. A protocol using LEDBAT ought to minimize the risk of such man-in-the-middle attacks by at least authenticating the timestamp field in the data packets and the delay field in the ACK packets.

LEDBAT is not known to introduce any new concerns with privacy, integrity, or other security issues for flows that use it. LEDBAT is compatible with use of IPsec and Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS).

7. 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.

8. References

8.1. 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.

[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.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

8.2. 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", October 2010, <http://arxiv.org/abs/1010.5623v1>.

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

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", RFC 6297, June 2011.

[Sch10] Schneider, J., Wagner, J., Winter, R., and H. Kolbe, "Out of my Way -- Evaluating Low Extra Delay Background Transport in an ADSL Access Network", Proceedings of 22nd International Teletraffic Congress (ITC22), September 2010.

[g114] "SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS; International telephone connections and circuits - General; Recommendations on the transmission quality for an entire international telephone connection; One-way transmission time", ITU-T Recommendation G.114, 05/2003.

[uTorrent] Hazel, G., "uTorrent Transport Protocol library", July 2012, <http://github.com/bittorrent/libutp>.

Appendix A. Measurement Errors

LEDBAT measures and uses one-way delays, and we now consider measurement errors in timestamp generation and use. In this section, we use the same locally linear clock model and the same terminology as Network Time Protocol (NTP) [RFC5905]. In particular, NTP uses the terms "offset" to refer to the difference between measured time and true time, and "skew" to refer to difference of clock rate from the true rate. A clock thus has two time measurement errors: a fixed offset from the true time, and a skew. We now consider these errors in the context of LEDBAT.

A.1. Clock Offset

The offset of the clocks, both the sender's and the receiver's, shows up as a fixed error in LEDBAT's one-way delay measurement. The offset in the measured one-way delay is simply the difference in offsets between the receiver's and the sender's clocks. LEDBAT, however, does not use this estimate directly, but uses the difference between the measured one-way delay and a measured base delay. Since the offset error (difference of clock offsets) is the same for the measured one-way delay and the base delay, the offsets cancel each other out in the queuing delay estimate, which LEDBAT uses for its window computations. Clock offset error thus has no impact on LEDBAT.

A.2. Clock Skew

Clock skew generally shows up as a linearly changing error in a time estimate. Similar to the offset, the skew of LEDBAT's one-way delay estimate is thus the difference between the two clocks' skews. Unlike the offset, however, skew does not cancel out when the queuing delay estimate is computed, since it causes the two clocks' offsets to change over time.

While the offset could be large, with some clocks off by minutes or even hours or more, skew is typically small. Typical skews of untrained clocks seem to be around 100-200 parts per million (ppm) [RFC5905], where a skew of 100 ppm translates to an error accumulation of 6 milliseconds per minute. This accumulation is limited in LEDBAT, since any error accumulation is limited to the amount of history maintained by the base delay estimator, as dictated by the BASE_HISTORY parameter. The effects of clock skew error on LEDBAT should generally be insignificant unless the skew is unusually high, or unless extreme values have been chosen for TARGET (extremely low) and BASE_HISTORY (extremely large). Nevertheless, we now consider the possible impact of skew on LEDBAT behavior.

Clock skew can manifest in two ways: the sender's clock can be faster than the receiver's clock, or the receiver's clock can be faster than the sender's clock. In the first case, the measured one-way delay will decrease as the sender's clock drifts forward. While this drift can lead to an artificially low estimate of the queueing delay, the drift should also lead to a lower base delay measurement, which consequently absorbs the erroneous reduction in the one-way delay estimates.

In the second case, the one-way delay estimate will artificially increase with time. This increase can reduce a LEDBAT flow's throughput unnecessarily. In this case, a skew correction mechanism can be beneficial.

We now discuss an example clock skew correction mechanism. In this example, the receiver sends back raw (sending and receiving) timestamps. Using this information, the sender can estimate one-way delays in both directions, and the sender can also compute and maintain an estimate of the base delay as would be observed by the receiver. If the sender detects the receiver reducing its estimate of the base delay, it may infer that this reduction is due to clock drift. The sender then compensates by increasing its base delay estimate by the same amount. To apply this mechanism, timestamps need to be transmitted in both directions.

We now outline a few other ideas that can be used for skew correction.

- Skew correction with faster virtual clock:

  Since having a faster clock on the sender will result in continuous updates of the base delay, a faster virtual clock can be used for sender timestamping. This virtual clock can be computed from the default machine clock through a linear transformation. For instance, with a 500 ppm speed-up the sender's clock is very likely to be faster than a receiver's clock. Consequently, LEDBAT will benefit from the implicit correction when updating the base delay.

- Skew correction with estimating drift:

  A LEDBAT sender maintains a history of base delay minima. This history 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, a host calculates 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 corrects fast, since state is preserved between connections.

Authors' Addresses

Stanislav Shalunov
BitTorrent, Inc.
303 Second St., Suite S200
San Francisco, CA 94107
USA

EMail: shalunov@shlang.com
URI: http://shlang.com

Greg Hazel
BitTorrent, Inc.
303 Second St., Suite S200
San Francisco, CA 94107
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