LEDBAT WG
Internet-Draft
Intended status: Experimental
Expires: January 13, 2011

S. Shalunov
G. Hazel
BitTorrent Inc
July 12, 2010

Low Extra Delay Background Transport (LEDBAT)
draft-ietf-ledbat-congestion-02.txt

Abstract

LEDBAT is an alternative experimental congestion control algorithm.

LEDBAT enables a 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 uTorrent, which is currently the most popular BitTorrent client, as the preferred congestion control mechanism, and deployed in the wild with favorable results.

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 January 13, 2011.

Copyright Notice

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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

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 first with the BitTorrent DNA client (a P2P-based CDN) and now 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

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 applicability

While originally deployed with a P2P application and a proprietary UDP framing, LEDBAT is applicable to any transport protocol that has provisions for its deployment and satisfies the wire format constraints (can carry timestamps, delay estimates, and has frequent ACKing). Currently, no IETF standard transport specifies the use of LEDBAT, but all of TCP, DCCP, and SCTP could be adapted to use it.

LEDBAT has been tested on the Internet across a range of link speeds from kilobits to multiple gigabits per second and in, given its deployment in uTorrent, across a very wide range of network topologies. It is designed to scale to higher speeds than the speeds tested, which were the highest the available hardware would support.

On the low end of the link speed spectrum, LEDBAT works best with slower links when the packet size is reduced, thus reducing serialization delay. On slower rural and wireless links, even a single 1500-byte packet can have a substantial serialization delay that, when combined with the protocol target delay, can become too large for VoIP applications sharing the bottleneck. The mechanism to reduce packet size for slower networks is not further discussed in this document.

5. LEDBAT motivation

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.

5.1. Simplest network topology

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.

5.2. Extra delay

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.

5.3. Queuing delay target

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.

5.4. Need to measure delay

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.

5.5. Queing delay estimate

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.

5.6. Controller

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.

5.7. 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.

5.8. Halve on loss

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. Note that, unlike with loss-based congestion control, the normal mode of LEDBAT operation does not induce losses and so normally does not rely on a loss rate to determine the sending rate. This makes the details of reaction to loss less important than in the case of loss-based congestion control. For LEDBAT, reducing the congestion window on loss is a safety net in case of severe congestion (or mistaken or broken delay estimates).

5.9. Yield to TCP

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.

5.10. Need for one-way delay

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.

5.11. 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.

5.12. Route changes

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).

When the connection is idle for a given minute, no data are available for the one-way delay and, therefore, no minimum is kept. When the connection has been idle for N minutes, the measurement thus has to begin anew.

5.13. 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.

5.13.1. 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.

5.13.2. Clock skew

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.

For cases when the base interval is long or the skew is high or the target is low, a technique to correct for skew can be beneficial. The technique described here or a different mechanism MAY be used by implementations. The technique described is still experimental, but it is actually currently used. The pseudocode in the specification below does not include any of the skew correction algorithms.

5.13.2.1. Deployed clock skew correction mechanism

Clock skew can be in two directions: either the sender's clock is faster than the receiver's, or vice versa. We refer to the former situation as clock drift "in sender's favor" and to the latter as clock drift "in receiver's favor".

When the clock drift is "in sender's favor", nothing special needs to be done, because the timestamp differences (i.e., the raw delay estimates) will grow smaller with time, and thus the base delay will be continuously updated with the drift.

When the clock drift is "in receiver's favor", the raw delay estimates will drift up with time, suppressing the throughput needlessly. This is the case that can benefit from a special mechanism. Assume symmetrical framing (i.e., same information about delays transmitted in both way). If the sender can detect the receiver reducing its base delay, it can infer that this is due to clock drift "in receiver's favor". This can be compensated for by increasing the sender's base delay by the same amount. Since, in our implementation, the receiver sends back the raw timestamp estimate, the sender can run the same base delay calculation algorithm it runs for itself for the receiver as well; when it reduces the inferred receiver's delay, it increases its own by the same amount.

5.13.2.2. Skew correction with faster virtual clock

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

Since having a faster clock on the sender is, relatively speaking, a non-problem, one can use two different virtual clocks on each LEDBAT implementation: use, for example, the default machine clock for situations where the instance is acting as a receiver, and use a virtual clock, easily computed from the default machine clock through a linear transformation, for situations where the instance is acting as a sender. Make the virtual clock, e.g., 500 PPM faster than the machine clock. Since 500 PPM is more than the variability of most clocks (plus or minus 100 PPM), any sender's clock is very likely to be faster than any receiver's clock, thus benefitting from the implicit correction of taking the minimum as the base delay.

Note that this approach is not compatible with the one described in the preceding section.

5.13.2.3. Skew correction with estimating drift

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

The history of base delay minima we already keep for each minute provides us with direct means of computing the clock skew difference between the two hosts. Namely, we can fit a linear function to the set of base delay estimates for each minute. The slope of the function is an estimate of the clock skew difference for the given pair of sender and receiver. Once the clock skew difference is estimated, it can be used to correct the clocks so that they advance at nearly the same rate. Namely, the clock needs to be corrected by half of the estimated skew amount, since the other half will be corrected by the other endpoint. Note that the skew differences are then maintained for each connection and the virtual clocks used with each connection can differ, since they do not attempt to estimate the skew with respect to the true time, but instead with respect to the other endpoint.

5.13.2.3.1. Byzantine skew correction

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

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 in the preceding section, then take the median of the skew differences.

This inherent clock drift can then be corrected with a linear transformation before the clock data is used in the algorithm from the preceding section, the currently deployed algorithm, or nearly any other skew correction algorithm.

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.

5.14. Noise filtering

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.

5.15. Non-bulk flows

Normally in transport, we're mainly concerned with bulk flows with infinite sources. Sometimes this model needs modification. For example, an application might be limiting the rate at which the data is fed to the transport layer. If the offered rate can be carried by the network without any sign of congestion, an adaptive congestion control loop will increase the rate allowed by congestion control. The LEDBAT mechanism in the form described above will keep increasing the congestion window linearly with time. This is clearly wrong for non-bulk flows because it allows the congestion control to decide that it is allowed to send much faster than it has ever sent. Therefore, some special treatment is required to deal with the case of non-bulk flows.

The same problem may also occur in some TCP implementations. It has first been noticed for TCP in situations where a connection has been idle for a substantial time and the original workaround was to re-enter slow start after an idle period. While re-entering slow start (or, more generally, returning to the original state of the connection) is a good idea after a long idle period, it does not solve the problem for connections that are trickling data out at an

application-determined rate. In fact, even application-level keepalives or small metadata exchanges can be enough to keep the connection from becoming idle for long. The fix for this problem is to limit the growth of the congestion window by a quantity related to the current flight size, i.e., the amount of outstanding unacknowledged data.

The congestion window should not grow beyond a factor of the current flight size for sufficiently large values of the flight size. This provides leeway for the congestion window to grow and probe the network beyond the current rate, at the same time keeping it on a tether that does not let it increase indefinitely. A refinement of this rule is that the congestion window should not grow beyond a constant plus a factor of the flight size. The presence of the additive constant is necessary because the congestion window needs to be able to increase at least by one packet even when it is small --

otherwise no probing is possible. The smallest value, one packet, then would define the minimum value of the factor: 2. However, for larger values of congestion window, a tighter bound than "double the flight size" may be desirable. The additive constant makes it possible to use values between 1 and 2 for the factor.

5.16. LEDBAT framing and wire format

The actual framing and wire format of the protocol(s) using the LEDBAT congestion control mechanism is outside of scope of this document, which only describes the congestion control part.

There is an implication of the need to use one-way delay from the sender to the receiver in the sender. An obvious way to support this is to use a framing that timestamps packets at the sender and conveys the measured one-way delay back to the sender in ack packets. This is the method we'll keep in mind for the purposes of exposition. Other methods are possible and valid.

Another implication is the receipt of frequent ACK packets. The exposition below assumes one ACK per data packet, but any reasonably small number of data packets per ACK will work as long as there is at least one ACK every round-trip time.

The protocols to which this congestion control mechanism is

applicable, with possible appropriate extensions, are TCP, SCTP, DCCP, etc. It is not a goal of this document to cover such applications. The mechanism can also be used with proprietary transport protocols, e.g., those built over UDP for P2P applications.

5.17. Fairness between LEDBAT flows

The design goals of LEDBAT center around the aggregate behavior of LEDBAT flows when they compete with standard TCP. It is also interesting how LEDBAT flows share bottleneck bandwidth when they only compete between themselves.

LEDBAT as described so far lacks a mechanism specifically designed to equalize utilization between these flows. The observed behavior of existing implementations indicates that a rough equalization, in fact, does occur.

The delay measurements used as control inputs by LEDBAT contain some amount of noise and errors. The linear controller converts this input noise into the same amount of output noise. The effect that this has is that the uncorrelated component of the noise between flows serves to randomly shuffle some amount of bandwidth between flows. The amount shuffled during each RTT is proportional to the

noise divided by the target delay. The random-walk trajectory of bandwidth utilized by each of the flows over time tends to the fair share. The timescales on which the rates become comparable are proportional to the target delay multiplied by the RTT and divided by the noise.

In complex real-life systems, the main concern is usually the reduction of the amount of noise, which is copious if not eliminated. In some circumstances, however, the measurements might be "too good" -- since the equalization timescale is inversely proportional to noise, perfect measurements would result in lack of convergence.

Under these circumstances, it may be beneficial to introduce some artificial randomness into the inputs (or, equivalently, outputs) of the controller. Note that most systems should not require this and should be primarily concerned with reducing, not adding, noise.

5.17.1. Late comers

With delay-based congestion control systems, there's a concern about the ability of late comers to measure the base delay correctly. Suppose a LEDBAT flow saturates a bottleneck; another LEDBAT flow starts and proceeds to measure the base delay and the current delay and to estimate the queuing delay. If the bottleneck always contains target delay worth of packets, the second flow would see the bottleneck as empty start building a second target delay worth of queue on top of the existing queue. The concern ("late comers' advantage") is that the initial flow would now back off because it sees the real delay and the late comer would use the whole capacity.

However, once the initial flow yields, the late comer immediately measures the true base delay and the two flows operate from the same (correct) inputs.

Additionally, in practice this concern is further alleviated by the burstiness of network traffic: all that's needed to measure the base delay is one small gap. These gaps can occur for a variety of reasons: the OS may delay the scheduling of the sending process until a time slice ends, the sending computer might be unusually busy for some number of milliseconds or tens of milliseconds, etc. If such a gap occurs while the late comer is starting, base delay is immediately correctly measured. With small number of flows, this appears to be the main mechanism of regulating the late comers' advantage.

5.18. Safety of LEDBAT

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.

6. LEDBAT congestion control

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 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
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 initialization:
  set all NOISE_FILTER delays used by current_delay() to +infinity
  set all BASE_HISTORY delays used by base_delay() to +infinity
  last_rollover = -infinity # More than a minute in the past.

on acknowledgement:
  delay = acknowledgement.delay
  update_base_delay(delay)
  update_current_delay(delay)
  queuing_delay = current_delay() - base_delay()
  off_target = TARGET - queuing_delay + random_input()
  cwnd += GAIN * off_target / cwnd
  # flight_size() is the amount of currently not acked data.
  max_allowed_cwnd = ALLOWED_INCREASE + TETHER*flight_size()
  cwnd = min(cwnd, max_allowed_cwnd)

random_input()
  # random() is a PRNG between 0.0 and 1.0
  # NB: RANDOMNESS_AMOUNT is normally 0
  RANDOMNESS_AMOUNT * TARGET * ((random() - 0.5)*2)

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 100 milliseconds. GAIN SHOULD be set so that max rampup rate is the same as for TCP and MAY be up to twice more. BASE_HISTORY SHOULD be 10; it MUST be no less than 2 and SHOULD NOT be more than 20. NOISE_FILTER SHOULD be 1; it MAY be tuned so that it is at least 1 and no more than cwnd/2. ALLOWED_INCREASE SHOULD be 1 packet; it MUST be at least 1 packet and

SHOULD NOT be more than 3 packets. TETHER SHOULD be 1.5; it MUST be

greater than 1. RANDONMESS_AMOUNT SHOULD be 0; it MUST be between 0 and 0.1 inclusively.

Note, in particular, that using a consistent TARGET value is important, as flows using different TARGET values will not share a bottleneck well: flows with higher values will tend to get a disproportionately large share of the bottleneck. This is the reason why the TARGET is the only parameter that is specified with no flexibility in the implementation. (Note that historically uTorrent used different values of TARGET, both as a temporary safety precaution and later experimentally. The current choice is based on the experience with the use of different values.)

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. 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.

8. Normative References

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

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