Assessing LEDBAT's Delay Impact

David Ros and Michael Welzl

***Abstract*—The major goal of the LEDBAT congestion control mechanism is to limit the amount of queuing delay that it causes. Using simulations, we show two aspects of the mechanism that may sometimes make LEDBAT's delay impact quite significant: first, it updates its "base delay" every 10 minutes, but may measure its own induced delay in doing so, even in the presence of significant transmission jitter. This may lead to a continuous growth of the fixed delay that a LEDBAT flow produces if it runs for long enough. Second, it requires competing traffic to be long-lived and greedy for it to reduce its rate accordingly, however, such traffic is in general not latency-critical.**

***Index Terms*—Transport protocols; congestion control; quality of service; less-than-best effort service.**

I. Introduction

LOW Extra Delay Background Transport (LEDBAT) [8] is a new, experimental congestion control mechanism, defined by the Internet Engineering Task Force (IETF) in the working group of the same name[1]. Its goal is to provide a "scavenger", or "less than best effort", service to applications that have loose constraints in terms of data delivery deadlines or throughput. A LEDBAT sender adapts its sending rate according to forward queuing delay by inspecting Acknowledgements (ACKs) from the receiver. These ACKs convey the difference between local clocks at the receiver and the sender. By computing the difference between the minimum delay value observed over a long time interval (called *base delay*) and every new delay sample from ACKs, the sender can estimate the amount of "excess" delay above the base value—i.e., the queuing delay—, without the need for costly clock synchronization between sender and receiver.

With every acknowledgement, the congestion window `cwnd` is increased or decreased in proportion to the difference between measured queuing delay and a `TARGET` value, specified as $\leq$ 100 ms in [8]. Window increases are limited so that they never are larger than those of a TCP flow, in similar circumstances; moreover, the closer the queuing delay is to `TARGET`, the slower `cwnd` increases. In case losses are detected, a LEDBAT sender reverts to a TCP-like behavior.[2]

Manuscript received January 17, 2013. The associate editor coordinating the review of this letter and approving it for publication was A. Vinel.

This work was partially funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

D. Ros is with the Institut Mines-Télécom / Télécom Bretagne, Rue de la Châtaigneraie, CS 17607, 35576 Cesson Sévigné cedex, France (e-mail: david.ros@telecom-bretagne.eu).

M. Welzl is with the University of Oslo, Department of Informatics, PO Box 1080 Blindern, N-0316 Oslo, Norway (e-mail: michawe@ifi.uio.no).

Digital Object Identifier 10.1109/LCOMM.2013.040213.130137

[1]http://tools.ietf.org/wg/ledbat

[2]Due to space limitations, we refer the reader to [5] for a more detailed exposition of how LEDBAT works.

There have been quite a few papers devoted to studying the performance of LEDBAT (see [5] and references therein). Several authors (e.g., [2]) have pointed to potential problems with the current LEDBAT specification. However, somewhat surprisingly, almost all of those works have focused on issues like efficiency and intra- and inter-protocol fairness, in terms of throughput, while little attention has been paid to *the impact that LEDBAT might have on the latency experienced by other flows*. To the best of our knowledge, Schneider et al. [7] is the only paper that explicitly, though briefly, deals with delay and jitter caused by LEDBAT to latency-sensitive flows; besides, some related issues were presented by R. Jesup at the 84th IETF meeting [4]. Abu and Gordon [1] hint at possible latency-related issues, but they look only at the impact of large variations in end-to-end delay due to rerouting, and do not directly study the consequences on such flows. The official specification of the mechanism [8] does discuss some potential problems, but they are assumed to become non-issues in real settings, without readily-available experimental data to back up such claims—a LEDBAT-like mechanism has been deployed in peer-to-peer clients across the Internet, but so far there is only anecdotal evidence on its performance.

Our results suggest that caution should be taken, and that more experiments—and, possibly, some changes to the specification—are needed before LEDBAT can be considered innocuous to other flows in terms of delay. One important point is that LEDBAT is supposed to be particularly useful where bufferbloat [3] is present, since it attempts to keep delay bounded down to a "reasonable" value. However, as we will see, it is in scenarios with excess buffering that some of the potential troubles with LEDBAT may indeed arise.

II. Issues with LEDBAT: an assessment

In this section we discuss two potential issues with the current LEDBAT specification. Our focus is mainly on the impact that LEDBAT may have in terms of delay, and how it might degrade the quality of experience for latency-sensitive applications. We explore one possible mitigation to one of these problems ([8] assumes this mitigation arises naturally in real end-hosts), to see whether it may actually bring a benefit.

For our assessment of LEDBAT, we have followed a simulation-based approach. To avoid the common pitfalls of such an approach, we have tuned our simulation scenarios and code to try to mimic real-world effects, such as random fluctuations in inter-packet transmission times or operating system scheduling times. It is worth noting that other studies have found that e.g. unfairness due to latecomer-advantage issues arise not only in simulated environments but also in real testbeds [7]. Moreover, simulations allow us to vary the possible fluctuations of processing delay in hosts, giving us an even better grip on operational boundaries than some real-life tests can do, as such tests are inevitably bound to the equipment that is used.

We have adapted to ns-2 the LEDBAT code for Linux developed by Valenti et al. [6]; such code is publicly-available[3] and is based on the LEDBAT specification in [8]. It runs in ns-2 on top of the *TCP-Linux* framework [9]; this allows us to use a simulation model of LEDBAT's congestion control algorithms that is reasonably close to a real-world implementation. The code runs as a TCP congestion-control module, which is one of the possible implementation approaches suggested in [8].

A. Simulation scenarios

Unless stated otherwise, the simulated scenario tries to mimic as much as possible the one used in [7], which was implemented in an ADSL testbed:

- Dumbbell topology, with links between both senders and receivers to their respective next-hop access routers.
- The bottleneck link emulates an ADSL link, with 10 Mbit/s downlink and 1 MBit/s uplink speeds. LEDBAT data packets are sent in the *uplink* direction. All data packets are 1500-bytes long.
- The uplink bottleneck buffer is *bloated* (it can hold 32 packets, i.e., $\approx 4\times$ the bandwidth-delay product as seen by the highest-RTT flow). The maximum delay that this buffer can introduce is 384 ms, and LEDBAT is configured with `TARGET` = 100 ms ([8] points to anecdotal evidence for this value to work well), so one LEDBAT flow alone may keep a steady 100-ms queue without suffering any losses.

B. Base delay growth

According to [8], the base delay (used to compute forward queuing delay) has to be estimated as follows. Minimum delays are measured over one-minute intervals, and a sliding window of `BASE_HISTORY` such minimum values is kept; base delay is then taken as the lowest of the `BASE_HISTORY` minimum-delay samples. The goal of the sliding window is to let the sender "forget" old values that may no longer be valid due to, say, route changes. [8] recommends to set `BASE_HISTORY` = 10 (i.e., the history of minimum observed delays goes back 10 min), and this is the value we use in our assessment below.

Fig. 1 illustrates a potential problem with the base delay mechanism, viz., the sender may lose track of the *actual* minimum delay after `BASE_HISTORY` minutes. Assume that the LEDBAT sender finds an empty queue when it starts sending data. If the queue never fully empties (which is what happens in this deterministic, simple scenario), once the base-delay window drops the first delay sample, the sender "sees" its own induced delay (= `TARGET` ms) and takes it as the new base delay, then adding 100 ms of queued packets on top of it. After every successive `BASE_HISTORY`-min period, the same phenomenon happens again, but this time the apparent minimum delay is yet higher (an extra `TARGET` ms on top of the previous value), until the buffer is filled (at around $t =$ 30 min) and losses cause the sender to reduce its rate, allowing the buffer to be partially drained.

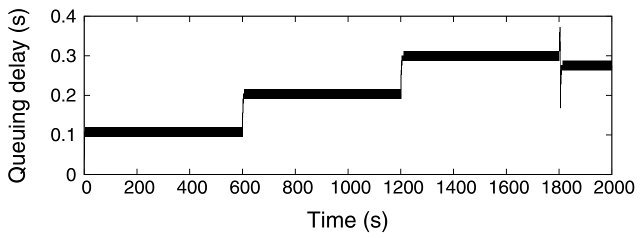


Fig. 1. Queuing delay increases every `BASE_HISTORY` = 10 minutes. There is no randomness in inter-packet transmission times, nor any transmission interruptions.

This is so because the LEDBAT flow is not able to sample the real minimum queuing delay (zero, in this case) if the flow keeps on sending steadily—i.e., if the queue never empties. The specification [8] *does* mention this issue, but dismisses it as follows: "We note that LEDBAT assumes *random fluctuations in inter-packet transmission times*. That will help to measure the correct base delay because the bottleneck runs empty from time to time." (emphasis added).

Are such random fluctuations in inter-packet transmission times enough to avoid what happens in Fig. 1? In principle, yes—*but they have to be very large*, i.e., on the order of the target delay or more. To test this, we repeated the same experiment, but adding *two* sources of randomness to the sending of data packets:

- *Jitter* at the sender. That is, the transmission of *every* packet is delayed by a random amount, drawn from a uniform distribution in $[0, \delta_{\max}]$ ms.
- *Transmission interruptions* that shut off at random the application on top of the LEDBAT sender, to resume later. The idea is to emulate e.g. disk-access delays, OS task scheduling, and the like. Interruptions happen at instants picked from a uniform distribution between 5 and 15 s (average = 10 s), and their duration is uniformly distributed between 0 and 50 ms[4].

Fig. 2 shows how the base delay evolves over time, for different values of $\delta_{\max}$ (picked as multiples of 12 ms, the transmission time of a 1500-byte packet at 1 Mbit/s). Remark that even fairly large gaps are *not* enough to ensure the base delay does not grow; nor are the longer-spaced transmission interruptions alone. In order for packet jitter to systematically allow for accurate sampling of the actual minimum delay, average inter-packet gaps $\delta_{\max}/2$ must be *at least as large as* $\approx$ `TARGET` ms—i.e., the sender has to stop sending for at least `TARGET` ms to ensure that the queue drains. Using a larger queue also means using more capacity; we therefore also examined the throughput of the flows depicted in Fig. 2, but we found that the impact of growing base delay on throughput was negligible.

C. Impact on short TCP flows

A main assumption behind LEDBAT is that, due to its reaction to delay, it will behave in a way that is unobtrusive to

[3]https://github.com/silviov/TCP-LEDBAT

[4]We do not claim that this model accurately captures an actual end-host's behavior, only that it is an easy way of simulating such interruptions and of adding a longer-term source of randomness to the delay sampling process.

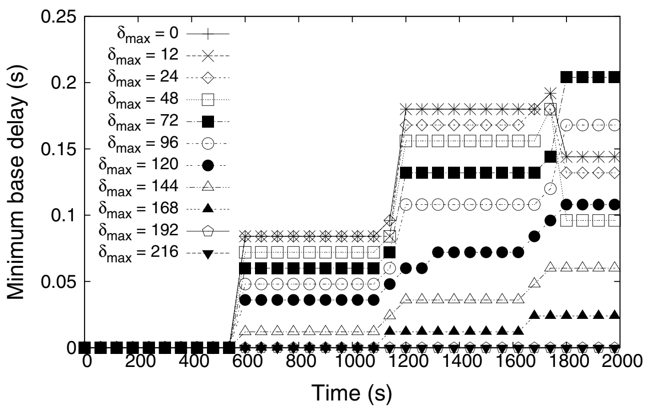


Fig. 2. Evolution of the minimum base delay over time. Inter-packet transmission times are uniformly distributed in $[0, \delta_{\max}]$ milliseconds.

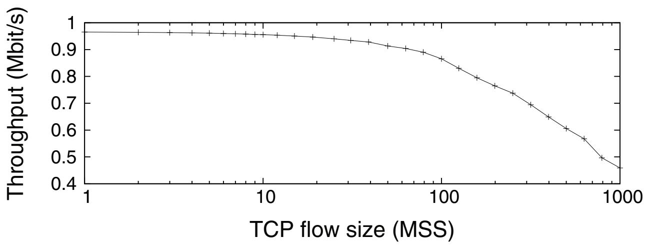


Fig. 3. Average throughput of the competing LEDBAT flow, for the case shown in Figure 4.

standard TCP flows. However, for a LEDBAT sender to yield to other flows, the latter have to really drive the queue towards saturation, so the former will manage to detect and react to such delay increases. That is, a competing TCP flow has to both last "long enough" and induce "enough extra delay" for a LEDBAT sender to be able to react in an appreciable way. If TCP traffic is composed essentially of short flows (web mice-like traffic), they will experience roughly an extra TARGET ms of delay, all the time. In other words: by trying to keep a standing queue, LEDBAT will tend to stay in the foreground, except if competing traffic is long-lived *and* greedy. *But long-lived, greedy traffic does not care about latency—it's short flows that care about it.*

To illustrate this issue, we simulated a TCP connection that sends data over the uplink, transferring over a long time a series of files of a given size. We measured transfer times[5] for different flow sizes (connection setup excluded), with and without a competing, long-lived LEDBAT flow. We also measured the average data rate achieved by the LEDBAT flow. For the TCP traffic, intervals between the end of a transfer and the start of the next transfer were picked at random, uniformly distributed between 5 and 15 s.

Figure 3 shows clearly that LEDBAT really yields only in the presence of large competing flows. This has a direct impact on transfer times for short flows. Figure 4 shows the resulting transfer times, as a function of file size (expressed in full-sized packets). Note how, for "small" flows (i.e., < 50 packets), transfer times with an ongoing LEDBAT flow are three to

[5]Defined as: the time elapsed between sending the first data packet of a file and receiving the ACK for the last data packet of that file.

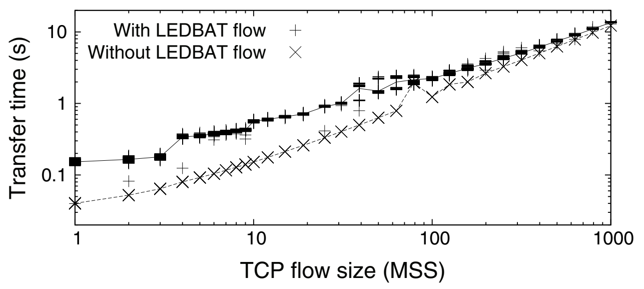
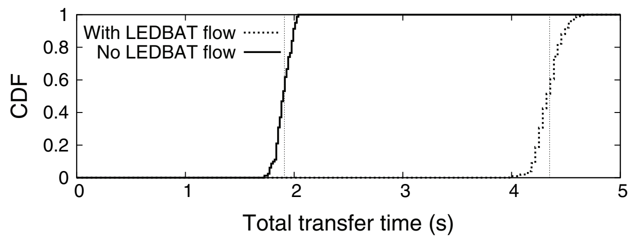
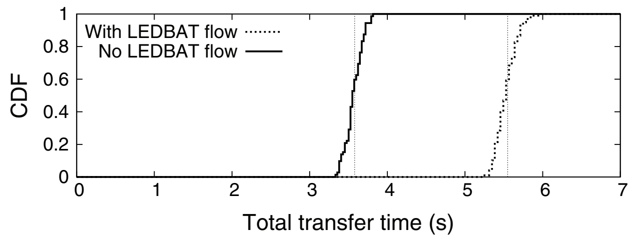


Fig. 4. Total transfer time for TCP flows of different sizes (in maximum segment size (MSS) units), with and without a competing LEDBAT flow. Each plot point corresponds to a measured transfer-time value; lines indicate average values. The RTT for the TCP flows is 25 ms.



(a) RTT for the TCP flows: 25 ms.



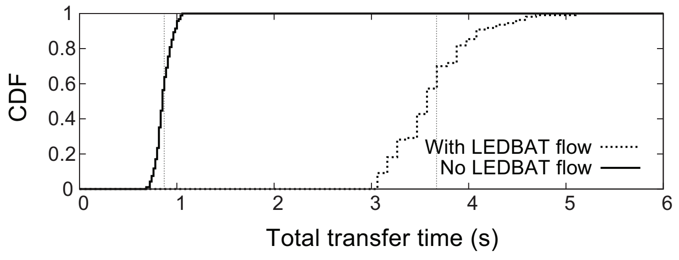
(b) RTT for the TCP flows: 100 ms.

Fig. 5. CDF of the total transfer time for web-like TCP flows, with and without a competing LEDBAT flow. The dotted vertical lines indicate average transfer times.

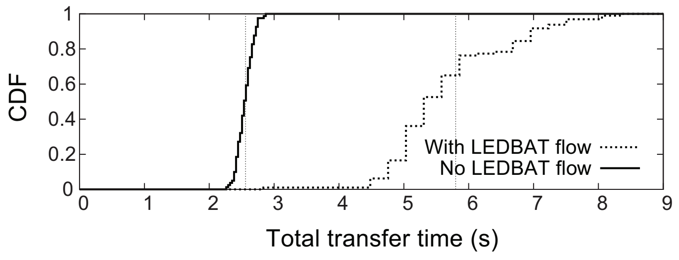
four times larger than the transfer time when there is no LEDBAT flow—e.g., for a 10-MSS TCP flow, average transfer time goes from 152 ms to 566 ms when the LEDBAT flow is added. When TCP flow sizes are large enough, however, the relative decrease in the extra latency is due to LEDBAT contributing less to delay (i.e., yielding more). Also, for larger flows, timeouts start to have an impact on transfer times, even when there is no LEDBAT flow.

The absolute increase in latency with a competing LEDBAT flow may seem small even if there is a large relative increase. However, many web flows today are composed of a *series* of short chunks of data (e.g., http messages sent in sequence, as discussed by Wischik [10]; in spite of improvements like the use of persistent connections, this traffic is still highly prevalent among web applications). In such a case, even a relatively small amount of additional *per-packet* latency will result in a *transaction* latency that is much higher.

We simulated such a sequence of web mice, as follows. A small chunk of data is sent, then there is a short wait, and then another chunk is sent; after 10 chunks have been sent and fully ACKed, there is a long "thinking" time (drawn from a uniform distribution, between 5 and 15 s) before the next



(a) RTT for the TCP flows: 25 ms.



(b) RTT for the TCP flows: 100 ms.

Fig. 6. CDF of the total transfer time for web-like TCP flows, with and without a "crossed" LEDBAT flow. The dotted vertical lines indicate average transfer times.

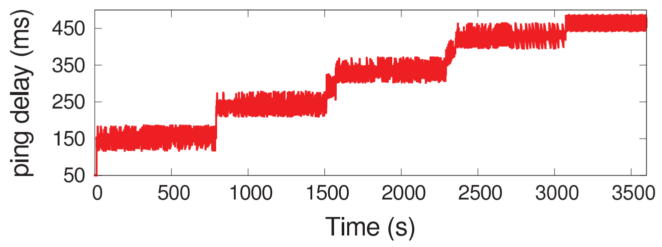


Fig. 7. Ping delay across an Ethernet link that was set to 500 kbit / 50 ms with netem, with the libutp's "utp_test" program running.

series of chunks starts. The size of chunks is drawn from a uniform distribution between 8 and 12 1500-byte packets. The short interval between two consecutive chunks in a "train" of 10 close chunks is drawn from a uniform distribution, between 0 and 5 ms.

The total transfer time measures the interval between starting sending the first chunk in a given series, and the end of reception of the last ACK for the last chunk in that series. The distribution of transfer times, depicted in Fig. 5, clearly show how the overall latency drastically increases in the presence of a long-lived LEDBAT flow. Naturally, the impact of the added TARGET ms of queue is relatively larger when the TCP flow has a lower minimum RTT.

Contrary to Fig. 5, in Fig. 6 web mice go in the *downstream* direction, while the LEDBAT flow still goes upstream. There is not as much competition with LEDBAT for the bottleneck buffer in the uplink (only TCP ACKs, not full-sized data packets). Still, by its contributing to increasing the RTT, the LEDBAT flow noticeably augments the overall latency.

III. Conclusions

The results presented here seem alarming. We only carried out simulations, and our models could be criticized—e.g., there are various ways to generate web-like traffic. Given that we have sketched situations in which LEDBAT does *not* work well, our results can however be regarded as "existence proof", and we note that there is probably reason to worry even if only a subset of them materializes in a practical setting. As mentioned earlier, other authors have found other issues with the mechanism. The oft-cited successful use of a LEDBAT variant in BitTorrent may not suffice to justify using LEDBAT for purposes such as backups or software updates: assuming default parameters, the latter involves continuous data transfers that may cause base delay to be refreshed multiple times, whereas further investigations would be needed to understand if the effects presented here may also appear with BitTorrent. To check if BitTorrent's underlying implementation itself may behave as in our simulations, we carried out a simple test in a LAN with BitTorrent's libutp[6], shown in Figure 7, where we can see a growing ping delay akin to the delay in Figure 1.

Stopping a transfer for at least TARGET ms before updating base delay, and choosing a smaller value for TARGET, could serve as simple temporary fixes to the problems that we have identified in this paper. We are not the first to say that 100 ms is a large value; we can however not make a more concrete recommendation at this stage. Picking the right value is a trade-off, the result of which should be evaluated in comparison with the related work in [5].

IV. Acknowledgements

D. Ros would like to express his gratitude to Prof. Stein Gjessing and the Department of Informatics (IFI) at the University of Oslo for their kind support.

References

[1] A. Abu and S. Gordon, "Impact of delay variability on LEDBAT performance," in *Proc. 2011 IEEE AINA*.
[2] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in *Proc. 2010 IEEE GLOBECOM*.
[3] J. Gettys, "Bufferbloat: dark buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 95–96, 2011.
[4] R. Jesup, "Issues with LEDBAT in wide deployment," in *2012 84th IETF Meeting*.
[5] D. Ros and M. Welzl, "Less-than-best-effort service: a survey of end-to-end approaches," *IEEE Commun. Surveys and Tutorials*, 2012, accepted for publication, to appear.
[6] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: the new BitTorrent congestion control protocol," in *Proc. 2010 ICCCN*.
[7] J. Schneider, J. Wagner, R. Winter, and H.-J. Kolbe, "Out of my way—evaluating Low Extra Delay Background Transport in an ADSL access network," in *Proc. 2010 ITC*.
[8] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, IETF, Dec. 2012.
[9] D. X. Wei and P. Cao, "NS-2 TCP-Linux: an NS-2 TCP implementation with congestion control algorithms from Linux," in *Proc. 2006 WNS2*.
[10] D. Wischik, "Short messages," *2007 Royal Society Workshop on Networks: Modelling and Control*.

[6]Downloaded from https://github.com/bittorrent/libutp on Dec. 3, 2012. The only code change was to multiply the variable "g_send_limit" by 10 in the file utp_test.cpp to transfer long enough.