Erratum: Evaluation of Different Decrease Schemes for LEDBAT Congestion Control

Mirja Kühlewind[1] and Stefan Fisches[2]

Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart, Germany
mirja.kuehlewind@ikr.uni-stuttgart.de
sfisches@ikr.uni-stuttgart.de

R. Lehnert (Ed.): EUNICE 2011, LNCS 6955, pp. 112–123, 2011.
© Springer-Verlag Berlin Heidelberg 2011

DOI 10.1007/978-3-642-23541-2_27

The original version of this article unfortunately contained a mistake in the presentation of the formula on page 116. The corrections are given below:

1) Instead of "$CWND = \frac{TARGET}{current_RTT} \times CWND$",

   it should read "$CWND = (1 - \frac{TARGET}{current_RTT}) \times CWND$"

2) The line above the formula "Thus β must be $\frac{TARGET}{current_RTT}$." should read

   "Thus the congestion window must be reduced by $\frac{TARGET}{current_RTT} \times CWND$ and

   β must be $1 - \frac{TARGET}{current_RTT}$."

The original online version for this chapter can be found at
http://dx.doi.org/10.1007/978-3-642-23541-2_13

Evaluation of Different Decrease Schemes for LEDBAT Congestion Control

Mirja Kühlewind[1] and Stefan Fisches[2]

[1] Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart, Germany
mirja.kuehlewind@ikr.uni-stuttgart.de
[2] sfisches@ikr.uni-stuttgart.de

Abstract. Low Extra Delay Background Transport (LEDBAT) is a new, delay-based congestion control algorithm that is currently under development in the IETF. LEDBAT has been proposed by BitTorrent for time-insensitive background traffic that otherwise would disturb foreground traffic like VoIP or video streaming. During previous evaluations the so called late-comer advantage has been discovered which makes a new starting LEDBAT flow predominant against already running LEDBAT flows. In this paper we evaluate different decrease schemes which have been proposed to solve this problem. We found that the proposed solutions come with a lower utilization, sometimes increased completion times and are much more sensitive to noise, which is contra-productive for the considered traffic class. Furthermore, we propose extensions to both evaluated schemes. We show that our approach can help to yield more quickly to higher priority traffic. We argue that a fair and equal share is not required for the specific traffic class LEDBAT is designed for. But it is important to address different application requirements in congestion control like LEDBAT as an approach for less-than-best effort background traffic.

1 Introduction

A substantial portion of bandwidth in today's Internet is used for background and time-insensitive traffic (e.g. P2P Traffic [1]). This traffic should not impede foreground and time-sensitive traffic.

A novel congestion control algorithm designed for less-than-best-effort traffic is the Low Extra Delay Background Transport (LEDBAT) [2]. It was proposed by BitTorrent in December 2008 and is now under development within an Internet Engineering Task Force (IETF) working group. It is a delay-based approach that can react earlier to congestion then loss-based schemes which are mostly used today in today's Internet for TCP traffic. In this paper, we regard two sorts of traffic:

1. Less-than-best-effort, low priority traffic using LEDBAT congestion control, e.g. automatic software Updates running in the background or Peer-to-Peer file sharing

R. Lehnert (Ed.): EUNICE 2011, LNCS 6955, pp. 108–119, 2011.
© Springer-Verlag Berlin Heidelberg 2011

2. Higher priority best-effort traffic using lost-based congestion control (or even sending with a constant bit rate), e.g. web-browsing or Voice over IP using UDP

According to [2], the LEDBAT congestion control seeks to:

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

With the current specification of LEDBAT there is a so-called "late-comer's advantage", where a second, newly starting flow can starve the first, already running one. This only happens when two or more LEDBAT flow compete on the same link and no other higher priority traffic is present. Thus it is an issue of intra-protocol fairness. Several mechanisms have been proposed to prevent this effect e.g. a mandatory slow-start or multiplicative decrease. We argue however that a high link utilization is actually more desirable for a lower priority traffic class than fairness within that class. As LEDBAT is designed for background traffic that will yield for higher priority traffic, LEDBAT should be able to utilize the link as much as possible when no other traffic is present. Completing one flow after another instead of transmitting all flows in parallel will, whilst being unfair, minimize the mean completion time. When sending as much data as possible en-block, computational power and hence energy consumption will be minimized as well.

To support our hypothesis we evaluate different decrease schemes with regard to completion time and utilization. We evaluated the proposed linear decrease scheme, which is discussed in the ITEF, and a contra-proposal by Carofiglio *et al.* [3] for using a multiplicative decrease. Moreover, we introduce new extensions to either of the schemes. We implemented LEDBAT as TCP congestion control in Linux using the TCP Timestamp Option for one-way delay measurements and subsequently used this code within a simulation environment. Moreover, we show that the decrease behavior is not only important to mitigate the effects of the late-comer's advantage but is also important when LEDBAT needs to yield to higher priority traffic like standard TCP.

The remainder of this paper is structured as follows: Section 2 summarizes related work. Section 3 is a general introduction into the LEDBAT algorithm and different decrease schemes. In Section 4 we present our TCP implementation in Linux. Section 5 shows our results regarding fairness and utilization.Section 6 gives concluding remarks.

2 Related Work

A comparison of LEDBAT with standard TCP as well as other less-than-best-effort congestion control mechanism such as LP-TCP and TCP-NICE has already been performed by Rossi *et al.* [4] [5]. They also detected the late comer's

advantage [6]. Initially, the authors proposed TCP Slow-Start as a solution to this problem. Slow-Start will usually overshoot and induce losses that cause all competing flows to basically restart their transmission. When two LEDBAT flows start at the same time, they will equally share the available capacity. But Slow-Start overshoot will actually affect all competing flows on the link, LEDBAT flows as well as higher priority standard TCP flows. This breaks one of the design goals of LEDBAT, as listed in 1. In fact, the current LEDBAT specification leaves it to the implementor to use a specific start-up scheme, if necessary.

[3] proposed a multiplicative decrease scheme to achieve fairness. We argue that equal sharing is a non-requirement for background traffic within its traffic class. [7] argues as well that it is not the right metric for fairness to share the available capacity equally between competing flows with different requirements.

Another study about parametrization is provided by [8]. In this paper we did not look at any parametrization issues. This issues are widely discussed on the IETF LEDBAT mailing list and mostly addressed in the working group document.

3 LEDBAT

LEDBAT is a novel delay-based congestion control approach for low priority transmissions. It is under development within an IETF working group. It is based on one-way delay measurements. When the measured one-way delay increases, LEDBAT can react earlier to congestion than loss-based approaches, which are more predominant in today's Internet. By slowing down its transmission rate earlier it will give the available bandwidth to presumed higher priority transmissions. Thus LEDBAT is friendly to most of today's higher priority TCP traffic.

Using timestamps, LEDBAT measures the queuing delay on a link. Assuming all queues on the path are empty at some point of the transmission, the sender will take the smallest delay measurement as the *base_delay*.

$$base_delay = min(base_delay, current_delay)$$

The *base_delay* is the constant fraction of the time a packet needs from sender to receiver and thus the minimum transmission time that can be observed. Any additional, variable delays are presumed to be waiting times in network queues. Therefore the actual queuing delay is calculated based on the current delay measurement given by the receiver as following:

$$queuing_delay = current_delay - base_delay$$

When the base delay changes during a transmission, e.g. because of re-routing, LEDBAT will automatically update to the new *base_delay* if it gets smaller. To recognize a higher base delay a base delay history is kept which holds the measured base delay of the last n minutes and will discard old values after $n+1$ minutes, thus adapting to the new base delay.

LEDBAT aims to keep the queuing delay low but not to be zero since optimal resource utilization requires that there is always data to send. Therefore LEDBAT

tries to achieve an extra delay of *TARGET* milliseconds. LEDBAT can determine how far off target it is and then increases or decreases its sending rate. LEDBAT is designed to not ramp up faster than TCP. Thus it will at maxium increase its sending rate by one packet per round-trip time (RTT). LEDBAT can use a filter to smooth or single out wrong delay measurements. But depending on the used filter scheme and length, e.g. minimum or average of the last *CURRENT_FILTER* samples, the reaction to congestion might get delayed.

3.1 Late-Comer's Advantage

Whenever a LEDBAT flow uses a previously unused path, it will immediately measure the real base delay. Thus it will saturate the bottleneck link after a short time by maintaining an extra delay of *TARGET* milliseconds. If now a second flow arrives, it can only measure the actual base delay plus the extra delay of the first flow. Wrongly, the second flow will take this value as its *base_delay*. While the latecomer will add its own target delay on top, the first flow measures an increased delay and begins to lower its sending rate. In the worst case, the second flow will add an additional *TARGET* millisecond of extra delay. As long as the second flow is not able to measure the actual base delay, it will push away the first flow completely. This effect is called the "late-comer's advantage".

A proposed way to mitigate this effect is to change the decrease behavior to multiplicative decrease to empty the queue completely such that the second flow can measure the real base delay [3].

3.2 Managing the Congestion Window

Linear Controller. The current draft version of LEDBAT uses the same linear controller for an additive increase and additive decrease.

$$off_target = \frac{TARGET - queuing_delay}{TARGET}$$

$$cwnd += GAIN * \frac{off_target}{cwnd}$$

The congestion window (CWND), which gives the number of packets that can be transmitted in one RTT, is altered by the normalized *off_target* parameter that can be positive or negative and thus determines how the CWND grows or shrinks. When *GAIN* is one, LEDBAT will at maximum speed up as quickly as standard TCP because *off_target* will always be smaller than 1 and reach its maximum value when *queuing_delay* is zero.

Unfortunately, this approach allows LEDBAT to decrease very slowly if e.g. just one millisecond of extra delay above the *TARGET* is measured. To be friendly to standard TCP traffic, LEDBAT should at least decrease as quickly as standard TCP is increasing. The latest version of the LEDBAT draft in the IEFT allows a different *GAIN* value for the decrease than for the increase. We propose to use

$$GAIN = TARGET * N$$

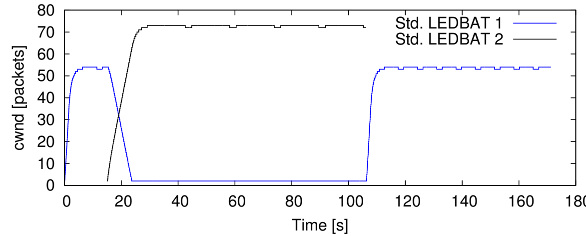

Fig. 1. Two competing LEDBAT flows with linear decrease

if *off_target* is negative (decrease). N would need to be the number of parallel starting standard TCP flows. As this number is usually unknown, we assume in most cases only one flow starting at the same time and set N to 1. Thus LEDBAT would decrease the congestion window at least by one packet per RTT.

Multiplicative Decrease. [3] proposes to decrease the congestion window for negative *off_target* by multiplying it with a factor β such that $0 < \beta < 1$:

$$CWND = \beta \times CWND$$

The idea is that the multiplicative decrease allows the queues to drain and thus enable a correct measurement of the base delay. If β is chosen to low it is underutilizing the link. If β is to large it doesn't drain the queue. While [3] is searching for a fixed value, β is actually depending on the chosen value of $TARGET$ in relation to the current RTT. This is because the CWND is depending on the RTT as it gives the number of packets that can be sent during one RTT. Whereas the $TARGET$ gives the extra delay and thus determines the number of packets that will be stored in the network queue that needs to be emptied at once (in one RTT). Thus β must be $\frac{TARGET}{current_RTT}$. For a negative *off_target* value the congestion window is then calculated as

$$CWND = \frac{TARGET}{current_RTT} \times CWND$$

$$current_RTT \approx 2 \times base_delay + TARGET$$

In our implementation we subtracted additional 3 packets after this calculation to encounter measurement and computation inaccuracies. This value of 3 packets is selected through simulative studies. With every multiplicative decrease scheme we decrease only once per RTT as during the first RTT after the decrease all delay measurements still reflect the situation before the decrease.

4 Linux Implementation

To provide a wide access to a less-than-best-effort congestion control scheme, we decided to implement LEDBAT as a TCP congestion control module. The Linux

kernel design provides an interface to extent the kernel functionality through additional modules. There is a specific module interface for congestion control. Thus our LEDBAT implementation follows the respective interface of the Linux kernel. The resulting c-file can be included in any current Linux kernel version. No further modifications were needed as TCP already provides an option to transfer and echo time-stamps. Based on this given functionality we do not calculate the delay at the receiver side, as specified in the LEDBAT draft, but at the sender. With the TCP Timestamp Option the receiver reflects the time-stamp $TSsnd$ sent by the sender and adds an additional time-stamp $TSrcv$ at sent-out of the ACK. By subtracting the echoed time-stamp (TSsnd) from the new time-stamp (TSrcv) we determine the one-way delay.

$$OWD = TSrcv - TSsnd$$

This delay includes the processing delay at the receiver, but as we assume this processing delay to be constant it will not disturb our queuing delay estimation. The clocks of sender and receiver do usually not operate on the same time base. As we only use *queuing_delay*, that means the variable part of the one-way delay in relation to a certain base delay, the absolute value of *base_delay* in not important for us. Even if both timestamp have a different resolution, this can be estimated by monitoring the first samples in relation to the own clock. As we know the time-stamp resolution in our evaluation scenarios, we did not implement a specific logic. In order to archive an precise enough delay measurements to monitor the changes in on-way delay, we had to adjust the kernels timer frequency from its default 250 HZ to 1000 HZ, giving us a 1 ms resolution in the timestamps.

Furthermore, TCP will acknowledge at least every second data packet and waits at most by default Linux configuration 100 ms for an additional packet. This is the delayed ACK mechanism of TCP. But whenever two packets are acknowledged at once, only the timestamp of the first packet will be echoed. Thus we have just half the number of measurement samples and some of them will have artifical delays. If packets arrive continuousely, there is a constant waiting time until the second packet arrives. This offset is not a problem. But if the timer expires, there are high variations. With the linear decrease scheme the impact of one wrong sample is quite low but with multiplicative decrease we necessarily need noise filtering to cope with this effect. For our simulations the open source character of Linux also allowed us to patch the kernel and disable delayed acknowledgments at the receiver side.

5 Evaluation Results

5.1 Scenario

To evaluate the presented decrease schemes in a real-world scenario we first set up a small testbed with the just described LEDBAT kernel module. All presented results are extracted from simulations with the IKR SimLib [9] as

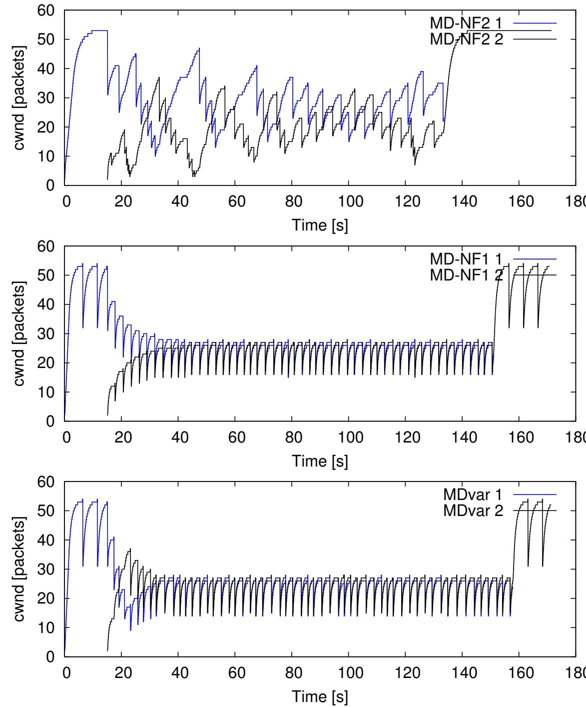

Fig. 2. Two competing LEDBAT flows with different multiplicative decrease, base RTT 40 ms

in the simulation delayed ACK could easily be deactivated. We used the IKR SimLib together with the TCP implementation of the real Linux kernel code provided by the Network Simulation Cradle [10] - a framework that makes kernel code usable within a simulation environment. In our simulation scenarios we used 2-5 parallel flows with a bottleneck link capacity of 10 Mbit/s and one-way delays of 10, 19, 20 or 30 ms. In every scenario each flow starts with an offset of 15 s to the previous one. All flows in a scenario want to transit the same data size which is either 30, 50, 100, 300, 500 or 1000 MBytes. The bottleneck node maintains a queue with a queue size that can hold 60 packets. For all simulations we used a *TARGET* value of 25 ms and a length *CURRENT_FILTER* of 1 or 2, where 1 basically means no filtering at all.

5.2 Two Competing Flows

To illustrate the behavior of the different decrease schemes we present in detail the scenarios where two LEDBAT flows compete with the same decrease mechanism. Figure 1 shows the linear decrease behavior. The first flow starts and increases its rate slowly until the bottleneck link is filled and 25 ms extra delay is introduced. The increase gets slower as it gets closer to the 25 ms target value. After 15 ms a second flow starts. This flow assumes the current delay as base

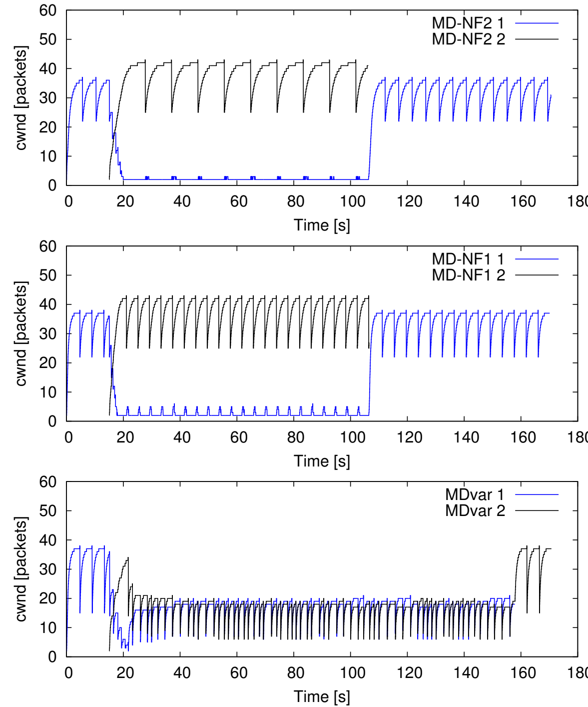

Fig. 3. Two competing LEDBAT flows with different multiplicative decrease, base RTT 20 ms

delay and starts increasing its rate as well. The first flows starts decreasing as it senses more extra delay than the already introduced 25 ms. It decreases only slowly as only little extra delay is introduced by the second flow so far. Thus the second flow, which never measures the correct base delay, adds an additional delay on top of the extra delay of the first flow. However, when two flows start at the same time or restart after the currently dominating flow finished, they will share the capacity equally as long as they have the same $TARGET$ value.

The upper and middle diagram in Figure 2 show the multiplicative decrease behavior as proposed by [3] and explained in 3.2 with a fixed value for β of 0.6. We investigated two different variants. The first variant uses a simple noise filter of length 2. It takes always the minimum of the last two measurement samples of *current_delay*. In the second variant the length is one, so there is no noise filtering. But we deactivated delayed ACKs in our simulation. We labeled those variants with *MD-NF2* and *MD-NF1*. All multiplicative decrease schemes aim to empty the queue when another LEDBAT is starting. Thus both flows can measure the right *base_delay*. Whenever some extra delay is introduced on top of TARGET these schemes will decrease the windows and the link becomes underutilized. This happens periodically even when no other flow is starting as LEDBAT itself will exceed the $TARGET$ for probing. When the second flow starts it will quickly measure the correct base delay as the first one decreases.

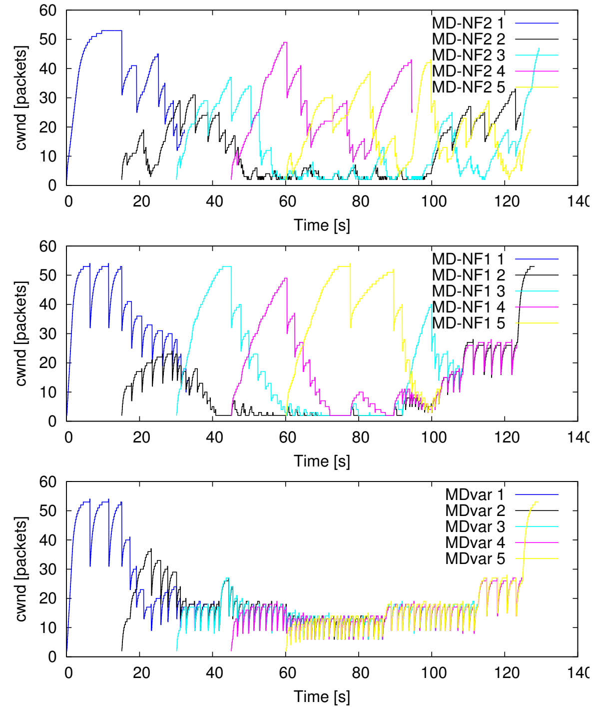

Fig. 4. Five competing LEDBAT flows with different multiplicative decrease, base RTT 40 ms

Then both will increase in parallel and the decrease times get synchronized so that both get the same share of the bandwidth. MD-NF2 does not synchronize due to the noise filtering and thus does not perfectly divide the bandwidth but approximately gives each flow an equal share.

The lower diagram in figure 2 shows our proposal with a dynamically adapted decrease factor (MDvar), deactivated delayed ACK and no noise filter. As both flows can have slightly different β values, they do not perfectly synchronize but work in all kinds of scenarios independent of RTT and number of competing flows as shown in Figure 2 (40 ms RTT) and 3 (20 ms RTT). Figure 4 shows the MD-NF1 scheme with 5 successively starting flows with an offset of 15 seconds. The first two do perfectly synchronize but all subsequent flows disturb the others before an equilibrium can be reached. With MD-NF2 flows are again not synchronized. In contrast the MDvar scheme shows that the flows always share the link equally after a short time.

Looking at transmission times we note that in Figures 1 and 2, no matter which decrease scheme, the last transmission to finish did so after 171 ms. With the linear decrease version shown in Figure 1 the second flow finishes its transmission ahead of the first flow after 106 ms. In all scenarios with multiplicative decrease in Figure 2 the first flow finishes after 134-158 ms which is nearly the same completion time than for the second flow (171 ms - 15 ms = 156 ms).

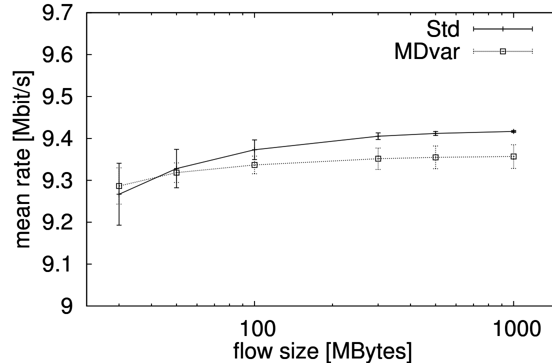

Fig. 5. Mean rate over flow size with 40 ms base RTT at 10 Mbit/s bottleneck link

Regarding bulk background traffic only the completion time of the whole transfer is relevant, not the instantaneous rate. In case of linear decrease at least one user will get a much better completion time whereas otherwise with the multiplicative decrease schemes all users have to wait quite long.

5.3 Utilization vs. Fairness

We argue that in the cases where LEDBAT should be used, like software updates in background, completion time and thus utilization is more important than an equal share of the capacity. Figure 5 shows the mean sum rate over different transfer sizes for each of the 2 to 5 parallel LEDBAT flows with a 15 s offset in the start time. In each scenario all competing flows have a minimum RTT of 40 ms on a common 10 Mbit/s bottleneck link. We only compared the standard LEDBAT with linear decrease and the MDvar scheme, as the other schemes do not work correctly in all scenarios regarding the fair share, as to be seen in Figure 3. The standard LEDBAT with linear decrease utilizes the link better with large transfers as each time a flow starts or ends, the link will not be fully utilized. With long transmissions these time periods in respect to the whole transmission get smaller.

We argue that the linear decrease scheme is more appropriate to LEDBAT traffic as completion time and utilization is most important. However, for background traffic, like automatic software updates, the completion time is not even important but it is important to not disturb other traffic. If we can utilize the link as much as possible when it is empty, we will prevent blocking any capacity for higher priority traffic that might arrive later-on. Moreover, the linear decrease scheme is less sensitive to noise and easier to implement.

5.4 Decrease Behavior with Competing Standard TCP Flows

Figure 5.3 shows one scenario that we found where LEDBAT with linear decrease is not friendly to standard TCP cross traffic. It is LEDBAT's most important design goal to yield quickly to higher priority TCP traffic, as described in 1. In

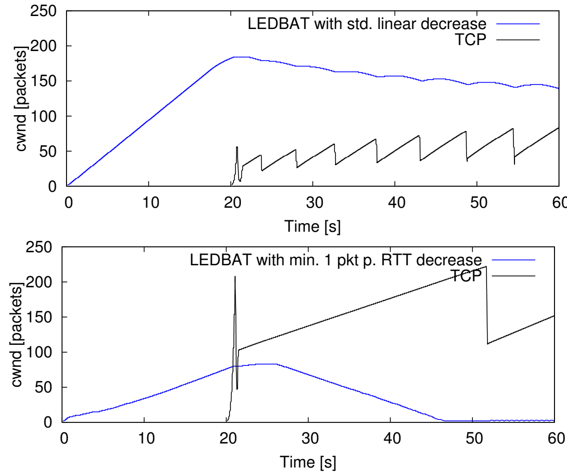

Fig. 6. LEDBAT flow standard linear decrease (upper) or with min. linear decrease of 1 packet per RTT (lower) and standard TCP cross traffic at 20 Mbit/s bottleneck link and queue size of 60 packets.

this scenario there is a bottleneck link of 20 Mbit/s and still the same queue size of 60 packets. To reach the *TARGET* of 25 ms the queue needs to be filled with 42 packets. 20 ms after the LEDBAT flow started, a standard TCP flow starts as well. After some RTTs LEDBAT started to decrease its sending rate. But it is only decreasing very slowly as it can at maximum sense another 11 ms of extra delay before the queue overflows. Unfortunately in this scenario, whenever the standard TCP fills up the queue and loss occurs, this loss only hits the standard TCP flow itself as it sends the data in bursts at the beginning of each RTT in Slow Start [11]. In this scenario the ratio between the TARGET and the maximum queue is very unfavorable. Moreover, the burst-wise transmission precludes LEDBAT from falling back into standard TCP behavior as it would do when loss occurs. We evaluated the same scenario with our changed approach where LEDBAT will decrease its rate at least by one packet per RTT. In Figure 6 we can see LEDBAT is yielding again to the TCP flow. In both of these scenarios delayed ACKs are not deactivated. With our proposal also the increase is slower than in Figure 5.3 as it decreases (more strongly) from time to time because of noise from delayed ACKs. An appropriate noise filter can help this problem, and thus we conclude a larger GAIN value for the decrease will help LEDBAT to reach its goals. An even larger decrease might be needed if multiple TCP flows simultaneously are sending as their sending rate will sum up to a larger increase than one packet per RTT.

6 Conclusion and Outlook

From our experiments we can see that while the standard linear decrease mechanism of LEDBAT always privileges the flow which started last, it is able to fully

utilize the link in stable state. Moreover, we recommend a larger GAIN for the decrease case to yield more quickly to competing higher priority standard TCP traffic. The latest version of the LEDBAT draft allows a higher GAIN for the decrease but it does not specify a certain value. We made a proposal to achieve a minimum decrease of one packet per RTT. Multiplicative decrease schemes, even our optimized proposal, in contrast, underutilize the link due to the periodical events where the queue is emptied. Given that LEDBAT is designed for lower-than-best-effort traffic, there is no demand for fairness but high link utilization and block-wise transfers will help to minimize the mean completion time.

LEDBAT will reset *base_delay* periodically when one or more LEDBAT flows maintain a constant extra delay on the link. Depending on the length of the base history filter the capacity share may change. If only some data are left to finish the transmission, a flow could actively reset *base_delay*. Of course, this will only cause an effect if just LEDBAT flows are on the link and thus will not disturb higher priority traffic.

References

[1] Schulze, H., Mochalski, K.: Internet study 2008/2009. IPOQUE Report (2009)

[2] Shalunov: S., Hazel, G., Iyengar, J., Kuehlewind, M.: Low extra delay background transport (LEDBAT). draft-ietf-ledbat-congestion-06 (2011)

[3] Carofiglio, G., Muscariello, L., Rossi, D., Valenti, S.: The quest for LEDBAT fairness. In: IEEE Globecom (December 2010)

[4] Rossi, D., Testa, C., Valenti, S.: Yes, we LEDBAT: Playing with the new bit-Torrent congestion control algorithm. In: Krishnamurthy, A., Plattner, B. (eds.) PAM 2010. LNCS, vol. 6032, pp. 31–40. Springer, Heidelberg (2010)

[5] Carofiglio, G., Muscariello, L., Rossi, D., Testa, C.: A hands-on assessment of transport protocols with lower than best effort priority. In: 35th IEEE Conference on Local Computer Networks, LCN 2010 (October 2010)

[6] Rossi, D., Testa, C., Valenti, S., Muscariello, L.: LEDBAT: the new BitTorrent congestion control protocol. In: International Conference on Computer Communication Networks, ICCCN 2010 (August 2010)

[7] Briscoe, B.: A fairer, faster internet protocol. IEEE Spectrum, 38–43 (2008)

[8] Schneider, J., Wagner, J., Winter, R., Kolbe, H.: Out of my Way Evaluating Low Extra Delay Background Transport in an ADSL Access Network. In: Proceedings of the 22nd International Teletraffic Congress (ITC22), pp. 7–9 (2010)

[9] IKR Simulation and Emulation Library, `http://www.ikr.uni-stuttgart.de/content/ikrsimlib/`

[10] Jansen, S., McGregor, A.: Simulation with Real World Network Stacks. In: Proc. Winter Simulation Conference, pp. 2454–2463 (September 2005)

[11] Allman, M., Paxson, V., Blanton, E.: TCP Congestion Control. RFC 5681 (2009)