Computer Networks 212 (2022) 109036

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

An experimental evaluation of LEDBAT++

Marcelo Bagnulo, Alberto García-Martínez [*]

Dep. Ing. Telemática UC3M, Spain

ARTICLE INFO

Keywords:
Transport
Congestion control
LEDBAT
Less-than-best-effort
Performance evaluation

ABSTRACT

LEDBAT++ is the evolution of LEDBAT, a congestion control algorithm originally designed to provide less-than-best-effort transport on the Internet. LEDBAT++ aims to address a number of shortcomings present in LEDBAT, including late-comer advantage, latency drift, competition on equal grounds with best effort traffic in the presence of small buffers and difficulties experienced while measuring the variations on the delay.

In this paper, we perform an experimental evaluation of LEDBAT++ using the Windows Server's LEDBAT++ implementation. We find that while LEDBAT++ overcomes all the limitations identified in LEDBAT, the change introduced in LEDBAT++ to do so results in a performance penalty that prevents LEDBAT++ flows to seize all the available capacity when there is no competing traffic. We propose two simple modifications to the LEDBAT++ algorithm that would address the identified issues and reduce the penalty.

1. Introduction

LEDBAT (Low Extra Delay Background Transport) [1] is a congestion-control algorithm that implements a less-than-best-effort (LBE) traffic class. When LEDBAT traffic shares a bottleneck with one or more TCP connections using standard congestion control algorithms such as Cubic [2] (hereafter *standard-TCP* for short), it reduces its sending rate earlier and more aggressively than standard-TCP congestion control, allowing standard-TCP traffic to use more of the available capacity. This effectively implements an LBE traffic class that has less priority than standard-TCP/best effort traffic. In the absence of competing standard-TCP traffic, LEDBAT aims to make an efficient use of the available capacity, while keeping the queuing delay within predefined bounds. LEDBAT is currently used across the Internet to carry delay-insensitive background traffic, notably by Bittorrent for peer-to-peer file sharing [3]) and by Apple for software updates [4].

LEDBAT reacts both to packet loss and to variations in delay. Regarding to packet loss, LEDBAT reacts with a multiplicative decrease, similar to most TCP congestion controllers. Regarding to delay, LEDBAT aims for a target queueing delay. LEDBAT estimates the queueing delay as the difference between the base delay (i.e., the minimum delay observed throughout the duration of the flow) and the current delay. When the measured queueing delay is below the target, LEDBAT additively increases the sending rate and when the delay is above the target, it additively reduces the sending rate.

After 10 years of operational experience with LEDBAT, a number of limitations and shortcomings have been identified [1,5,6]. These include inter-LEDBAT fairness issues (notably, the so-called *late-comer advantage*), LEDBAT being overly aggressive when competing with standard-TCP in networks with small buffers, LEDBAT bloating the queues for long-lasting communications as well as multiple difficulties when it comes to measure the one-way delay LEDBAT uses to estimate the queueing delay.

LEDBAT++ [6] is the evolution of the original LEDBAT that includes modifications aimed to address the shortcomings identified in LEDBAT. Similarly to LEDBAT, LEDBAT++ reacts both to packet loss and to variations in delay. While the reaction to packet loss is unchanged in LEDBAT++, when it comes to reaction to delay variations, LEDBAT++ substitutes the additive increase/additive decrease mechanism of LEDBAT for an additive increase/multiplicative decrease (AIMD) in order to improve fairness. LEDBAT++ AIMD uses dynamic parameters that allows it to be less aggressive than standard-TCP even in networks with small buffers. Also, LEDBAT++ incorporates periodic slow-downs to allow flows to properly assess the base delay (i.e., the delay when there are no queues) and reduce both the late-comer advantage and the buffer bloating effects. LEDBAT++ uses Round Trip Times (RTTs) instead of one way delays for its calculations in order to avoid the difficulties with measuring the one-way delay.

Windows 10 anniversary update included LEDBAT++ support and from the 2019 version onwards, Windows Server also implements LEDBAT++ [7]. LEDBAT++ is currently being used by Microsoft software distribution servers (SCCM) to distribute software updates without interfering with other traffic while updating [8]. According to Microsoft,[1]

[*] Corresponding author.
E-mail addresses: marcelo@it.uc3m.es (M. Bagnulo), alberto@it.uc3m.es (A. García-Martínez).

[1] https://news.microsoft.com/bythenumbers/en/windowsdevices.

[2] https://www.catalog.update.microsoft.com/.

https://doi.org/10.1016/j.comnet.2022.109036
Received 29 December 2021; Received in revised form 28 April 2022; Accepted 8 May 2022
Available online 17 May 2022

there are 1.4 billion devices running Windows 10 or Windows 11. If we consider operating system updates only (and disregard updates for other software such as Office. etc.), Microsoft issues updates for Windows both monthly (the so-called "patches") and semi-annually (a.k.a. feature updates).[2] Updates regularly include fixes for bugs and security holes that are critical to maintain the Windows users and, by extension (due to the large installed base), the Internet as a whole, safe. Downloading a feature update using standard TCP can be quite disruptive for the users' communications, as it is a fairly large data transfer (about 1 GB of data) which competes in equal grounds with other user traffic. Through the use of LEDBAT++, if LEDBAT++ lives up to its promises, the disruption can be avoided and software can be updated seamlessly. It is then important to determine if LEDBAT++ actually exhibits all the features expected from an LBE transport.

In this paper, we perform an experimental evaluation of the LEDBAT++ congestion control algorithm. In particular, we aim to verify if LEDBAT++ overcomes the limitations of LEDBAT as expected by its design. We experimentally tested Windows Servers's LEDBAT++ implementation and performed over 2,000 experiments. We find that LEDBAT++ sufficiently achieves its design goals and addresses the LEDBAT issues described earlier. However, we also find that the mechanisms built into LEDBAT++ to do so impose a performance penalty that prevents LEDBAT++ from seizing all the available capacity when there is no competing traffic.

The rest of this paper is structured as follows. In Section 2 we describe the limitations of LEDBAT and in Section 3 we present how LEDBAT++ overcomes the described limitations of LEDBAT. In Section 4 we describe the setup used for our experiments. Next, in Sections 5–7 we present the results of our experiments regarding the performance of LEDBAT++ running solo in a bottleneck link, LEDBAT++ flows sharing a bottleneck link with standard-TCP and inter-LEDBAT++ fairness respectively. Section 8 describes the related work and Section 9 concludes the paper.

2. Limitations of LEDBAT

Since its inception, LEDBAT is known to suffer from the so-called *late-comer advantage* problem. As described in [1], LEDBAT offers more capacity to LEDBAT flows arriving later than to the LEDBAT flows already present in a bottleneck link. For example, if there is a LEDBAT flow using the whole capacity in a bottleneck link and later on another LEDBAT flow arrives, the second flow will expel the first flow and use all the available capacity. The reason for this is that the measure of the base delay by the second flow includes the queueing delay caused by the packets of the first flow already present in the bottleneck buffer. After this initial measurement, the second flow's LEDBAT controller aims to reach a target queueing delay in addition to the delay caused by the first flow, resulting in a higher delay than the one the first flow is aiming for, causing the first flow to reduce its sending rate to the minimum. The result is that the second flow uses all the available capacity.

This is not the only fairness issue that affects LEDBAT. LEDBAT also has poor fairness for multiple LEDBAT flows that start at the same time. As described in [5], this is essentially due to the additive increase/additive decrease algorithm used by LEDBAT, that converges to a stable split of capacities between flows, but with an equilibrium point that is not necessarily fair.

Other identified shortcomings are related to the difficulties measuring one-way delay in TCP. LEBDAT uses the differences on the measured one-way delays to estimate the queueing delay and react accordingly, depending whether the measured queueing delay is above or below the target. One-way delay can be measured in TCP using the TimeStamp option [9]. The problem is that one endpoint of the TCP connection does not know the units used by the other endpoint to express the TimeStamp value. Even after discovering the units of the other's endpoint TimeStamp values, the obtained values still need to be corrected for clock skew and drift, which is challenging.

It has also been pointed out that when the bottleneck link buffer is small, the measured queueing delay never reaches the target delay, rendering the delay-based mechanisms of LEDBAT irresponsive. In such situation, LEDBAT relies in its loss-based mechanisms that are exactly the same as standard TCP, losing its LBE characteristic and competing in equal grounds with standard TCP.

Last, in order to accommodate for routing changes that result in changes in the path used for the packets of the LEDBAT communication and the potential changes in the associated base delay, LEDBAT "forgets" delay measurements older than a given time (10 mins). While this accommodates for routing changes, it creates another problem called *latency drift* [10]. Consider the case of a LEDBAT flow running solo in a bottleneck link. The only time the LEDBAT sender is able to measure the real base delay is at the beginning of the communication. For the rest of the lifetime of the communication LEDBAT itself bloats the queue so that the queuing delay matches the target. If the initial measurements are discarded, then the lowest delays observed are roughly the real base delay plus the target. After that point, LEDBAT aims for a target queuing delay on top of that, resulting in a real queueing delay of twice the target. Then, 10 min later, it will aim for a queueing delay of three times the target and so on, creating the latency drift effect.

3. Overview of LEDBAT++

LEDBAT++ introduces a number of modifications to address the limitations identified for the original LEDBAT mechanism.

LEDBAT++ controls the sending rate through the calculation of a congestion window, CW. LEDBAT++ updates the CW based on delay variations and packet loss. With respect to packet loss, LEDBAT++ reacts by reducing the CW to half of its value when a loss is detected.

With respect to delay variations, LEDBAT++ aims for a pre-defined queueing delay target, T (defined equal to 60 ms in the specification). LEDBAT++ continuously estimates the current queueing delay, q_d. If the current queueing delay q_d is larger than the target queuing delay T, LEDBAT++ multiplicatively decreases the congestion window. Conversely, if the delay is smaller than the target, LEDBAT++ additively increases the congestion window.

LEDBAT++ estimates the current queueing delay (q_d) by subtracting the base round-trip-time (RTT_b) from the current RTT (RTT_c). The base RTT is calculated as the minimum RTT observed in the last 10 min of the lifetime of the communication. The current delay is the last RTT measured in the communication. Both values are filtered to eliminate noise by taking the minimum of the last n values, n being at least 4. The current queueing delay is then calculated as:

$$q_d = RTT_c - RTT_b \tag{1}$$

LEDBAT++ defines the GAIN parameter as follows[3]:

$$GAIN = \frac{1}{min(16, CEIL(2 * \frac{T}{RTT_b}))} \tag{2}$$

For a T equal to 60 ms, this means that GAIN is equal to 1 for base RTTs larger than 120 ms, equal to 0.5 for base RTTs of 60 ms and equal to $\frac{1}{16}$ for RTTs smaller than 7.5 ms. As it will be described later, the GAIN parameter is used to make LEDBAT++ AIMD less aggressive than the one used by standard-TCP, even in cases where the buffers are small. The (unstated) assumption is that networks with small base RTTs are more likely to have shallow buffers (e.g., datacenter networks).

LEDBAT++ AIMD reacts to changes in the queueing delay by updating its CW as follows:

if $q_d < T$, then

$$CW_{n+1} = CW_n + GAIN \tag{3}$$

[3] CEIL(X) is defined as the smallest integer larger than X.

and if $q_d > T$, then

$$CW_{n+1} = CW_n + max(-\frac{CW_n}{2}, (GAIN - CW_n.(\frac{q_d}{T} - 1))) \quad (4)$$

with CW_n being the value of the congestion window computed at RTT n.

Eq. (3) defines the Additive Increase part. It basically states that the CW increases up to 1 MSS per RTT (being 1 MSS if the base RTT is 120 ms or larger, and less than that for smaller base RTTs). The purpose of this is to ensure that LEDBAT++ increases less aggressively than standard-TCP when the base RTT is less than 120 ms. This is especially important when the bottleneck link buffer is small and LEDBAT++ is solely decreasing its CW based on losses.

Eq. (4) describes the Multiplicative Decrease part. By using multiplicative decrease (instead of the additive decrease used in LEDBAT), LEDBAT++ aims to overcome the (un)fairness issues identified in LEDBAT. The multiplicative decrease factor depends on the ratio between the current queuing delay and the Target T, so that LEDBAT++ reacts softly to small excesses in the queueing delay, allowing a smooth operation around the target point. The multiplicative decrease is capped to half, to avoid starving LEDBAT++ in cases of spikes in the delays.

LEDBAT++ performs a slow-start increase at the beginning of the connection. LEDBAT++ slow-start is similar to the one of standard-TCP, in the sense that the CW increases exponentially. However, in order to be less aggressive than standard-TCP, instead of doubling the CW every RTT, LEDBAT++ multiples the CW by a factor of $2 \cdot GAIN$, $GAIN$ being always less than or equal to 1. LEDBAT++ exits the exponential growth of the initial-slow start when the measured queuing delay surpasses $\frac{3}{4}$ of the Target T, in order to avoid overshooting.

In addition, LEDBAT++ performs periodic slow-downs to obtain more accurate measurements of the base RTT and overcome the latecomer advantage identified in LEDBAT. This means that periodically, LEDBAT++ sets the CW to two $MSSs$ during two $RTTs$ and then performs a slow-start increase back to the CW value that it was using before the periodic decrease. An initial slow-down is performed 2 RTTs after exiting the initial slow-start. After that initial slow down, LEDBAT++ performs slow-downs periodically. If we call Tss the time that it takes for the slow-start to ramp back up, then LEDBAT++ performs the next periodic slow down after a period equal to $9 \cdot Tss$.

4. Experimental setup

In Fig. 1 we present the virtualized environment we use for the LEDBAT++ experiments. We are featuring a *dumbbell* topology. Albeit this is a fairly simple topology with only 6 nodes, it is enough to thoroughly test a congestion control algorithm. As observed in the design of model-based congestion control algorithms, see [11], independently of the number of links a connection crosses, from the transport layer perspective, any path, no matter how complex it is, it behaves as a single link with the RTT of the overall path and the capacity of the path's bottleneck link, which is exactly what this simple topology represents.

C1, R1, R2 and S1 are Linux systems while S2 is a Windows 2019 Server with LEDBAT++ capability. LEDBAT++ traffic flows from S2 to C1 while CUBIC traffic flows from S1 to C1. Traffic is generated in S1 using the `nc` tool and in S2 using the `ctsTraffic` tool (i.e., bulk transfer type of traffic in both cases). The client in C1 uses `nc`. The link connecting R2 with R1 is the bottleneck link of the communications between S1 (S2) and C1. We set its capacity to different values using the `tc` traffic control tool. A drop-tail buffer is configured in the R2 to R1 link, with a size we may vary on different experiments, to represent different network setups. During the experiments, we also vary the RTT and the capacity of the bottleneck link. The links between S1 (S2) and R2 and the ones between C1 and R1 are configured with larger capacities than the one of the bottleneck. For all experiments, the link layer frame size is 1456 bytes and the MSS is 1390 bytes. In all the experiments, TCP flow control never limits the communication rate.

To compute the rates for each flow, we start a `tcpdump` capture in R1.

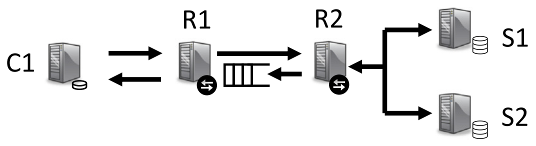


Fig. 1. Experiment setup.

5. LEDBAT++ solo performance

In this section, we analyze the performance of LEDBAT++ when there is no competing traffic in the bottleneck link, i.e., the performance of a single LEDBAT++ flow running solo in the bottleneck link. We consider different conditions varying the RTT and the bottleneck link capacity and analyze the resulting behavior.

LEDBAT++ has two different modes of operation, delay based and loss based. When the bottleneck link buffer has enough capacity to generate a queueing delay equal or higher than LEDBAT++'s target delay (T), then LEDBAT++ runs in delay based mode, while when the buffer is smaller, losses are generated before LEDBAT++ can react to the increase in the queueing delay, so it runs in loss based mode. We consider both modes of operation in our analysis.

5.1. LEDBAT++ solo performance in delay based mode

For this analysis, we configure a large buffer in the bottleneck link. Specifically, we configure a buffer of size B equal to 500 ms. We express the size of the buffer in milliseconds rather than in bytes. The size of the buffer in bits can be computed as $\frac{C*B}{1,000}$, with C being the bottleneck link capacity expressed in bps.

5.1.1. LEDBAT++ solo performance in delay based mode: variation with RTT

We perform experiments with a single LEDBAT++ flow running in a bottleneck link of capacity C equal to 20 Mbps, with a buffer of size B equal to 500 ms. We vary the base RTT (RTT_b) between 10 ms and 400 ms. Each experiment runs for 300 s and we compute the average rate of the LEDBAT++ flow for each experiment. We perform 8 runs for each RTT measured. The results are plotted in Fig. 2. The graph plots the rate achieved at the application layer.[4] The maximum application layer rate possible in a 20 Mbps link is also plotted.

By looking at the graph, we can observe that the rate achieved for small RTTs is close to the full capacity of the link, for medium RRTs, the rate decreases lightly and that for larger RTTs the rate decreases even further.

In order to gain some insights about the differences in performance depending on the base RTT, we look into the evolution of the flight-size (i.e., the number of bytes sent but not yet acknowledged) throughout the duration of particular experiments. We use the flight-size as an estimator of the sender window. We measure the flight-size to avoid instrumenting the Windows Server to retrieve the sender's window. In most cases, the sender window and flight-size matches (except when there are losses and mechanisms like Fast Recovery [12] are used, that they differ for the short period while the lost packet is recovered, which is not the case of LEDBAT++).

In Fig. 3 we plot the flight-size observed in one of the experiments with RTT set to 100 ms. Omitting the transient initial period, we observe that most of the time, the flight-size oscillates around the bandwidth-delay product (BDP) corresponding to a delay equal to

[4] The application level data accounts for 1390 B out of the 1456 B of the frame.

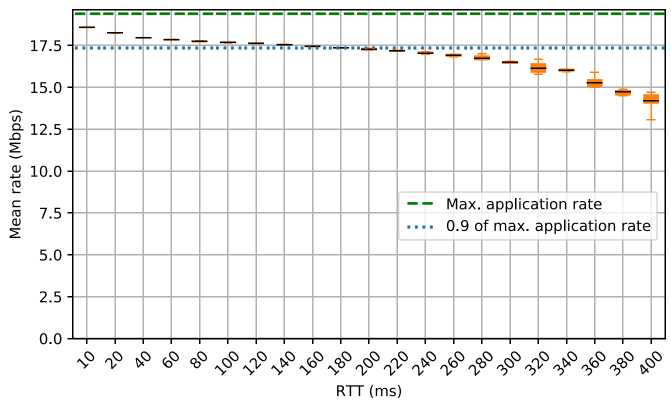


Fig. 2. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 500 ms, varying the base RTT.

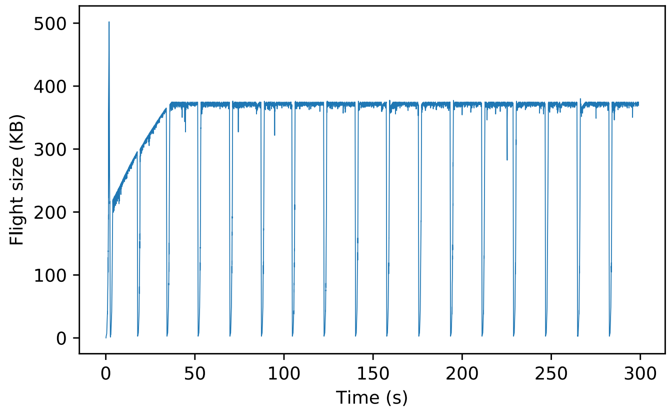


Fig. 3. Flight-size of a LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 500 ms, with base RTT set to 100 ms.

160 ms (i.e., 100 ms of RTT plus 60 ms of target T),[5] i.e., 372 kB. During the periods in which the flight-size is at the bandwidth delay product, full utilization of the bottleneck link capacity is achieved.

However, we can also observe that the periodic slowdowns reduce the flight-size, impacting the overall rate achieved. We can estimate the impact of the periodic slow downs as follows.

According to the specification, periodically, LEDBAT++ reduces the window to 2 MSS for 2 RTTs and then uses slow-start to increase the window back up to the value that it had before the periodic slow down. The behavior is repeated after 9 times the time that it took for slow-start to ramp back up the window. We observe this behavior in the plot, with the periodic reductions of the flight-size. As a rough approximation, for large enough BDPs (i.e., much larger than 2 MSSs), LEDBAT++ transmits at full speed during 9/10 of the time, while the remaining 1/10 of the time is (practically) not transmitting. So, naturally, LEDBAT++ achieves a rate of 9/10 of the available capacity since 1/10 of the capacity is wasted due to the periodic slow down. This is what is observed in Fig. 2 for RTTs between 20 ms and 300 ms where 0.9 of the maximum application layer rate is also plotted.

We present a more thorough justification of the $\frac{9}{10}^{th}$ approximation in Appendix.

For RTTs close to 0, we observe in Fig. 2 that LEDBAT++ is almost achieving the full utilization of the available capacity. This is so because the window needed to fill up the BDP is small. LEDBAT++ uses a minimum window of 2 MSS. For RTTs smaller than 1.2 ms, the full 20 Mbps of capacity are achieved using windows of 2 MSS or smaller. Also, for small RTTs, the number of rounds in exponential growth needed to restore the window is also small, so the penalty imposed by the slow-start is reduced. When the RTT increases, the window required to fill in the available capacity grows larger and the effect of the periodic slow down becomes more apparent, converging to the observed 9/10 of available capacity observed for larger RTTs. A more formal analysis of this is presented in Appendix.

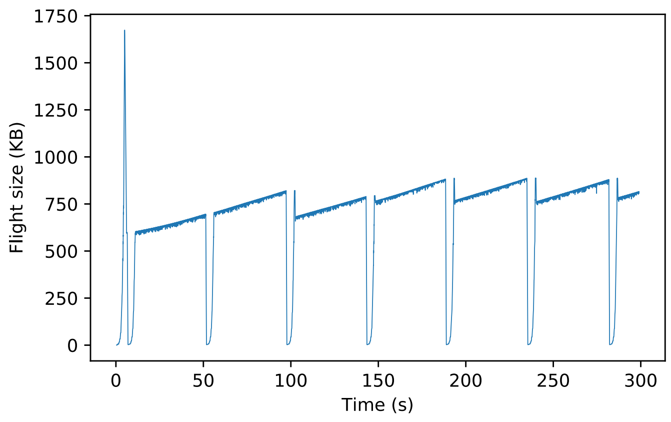


Fig. 4. Flight-size of a LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 500 ms, with base RTT set to 400 ms.

We next inspect what happens for larger RTTs, when the performance decreases beyond the 9/10 factor described before. In Fig. 4 we plot the flight-size of a single LEDBAT++ flow with an RTT of 400 ms. We observe that the maximum flight-size achieved is smaller than the one required to fill in the link. Specifically, for an RTT of 400 ms, the flight-size required to achieve 20 Mbps is 1 MB while the maximum flight-size observed on the graph is close to 800 kB.

We also observe that after a periodic slow down, LEDBAT++ attempts to restore the rate it was transmitting at before entering the slow down. However, after reaching the desired rate exponentially, it reduces its rate further and only then resumes the linear increase, until the next periodic slow down. We also observe that after 200 s, it enters in a steady state, and the growth during the linear increase matches the decrease suffered right after the exponential growth.

We next look at the observed RTTs, depicted in Fig. 5, to gain further insight. We can easily match the evolution of the measured RTT with the evolution of the rate observed in Fig. 4. The periods of linear increase of the rate match with the periods during which the RTT is close to the base RTT. During these periods, we observe that the RTT never reaches the base RTT plus the Target T (i.e., 460 ms) so never triggering the multiplicative decrease. We also observe periodic large peaks in the RTT. These peaks match with the exponential increases in Fig. 4. In these peaks, the measured RTTs surpass the 460 ms, resulting in a queuing delay that exceeds the target T of 60 ms, triggering the LEDBAT++'s multiplicative decrease which causes the reduction on the rate we observe right after each exponential growth in Fig. 4.

To understand what is causing these peaks in the measured RTT, we recall that during exponential growth, each ACK received generates GAIN additional packets. If RTT_b is larger than 120 ms, GAIN is equal to 1. This means that if in the nth RTT k packets were sent, in the $(n+1)^{th}$ RTT, 2k packets are issued. Because of ACK clocking, 2k packets will be issued during the time it takes the bottleneck to transmit k packets. So, packets spend some time in the buffer. If the queueing delay caused by these packets in the last round of the exponential growth exceeds the target T, it triggers a subsequent window reduction. After this window reduction, LEDBAT++ increases the window linearly, but only until the next periodic slow down, at which point the behavior repeats itself.

[5] Similarly to the previous plots, the values in Fig. 3 correspond to the application level data, which are 1390 B out of the 1456 B of the segments, so the appropriate correction factor should be included for the calculations.

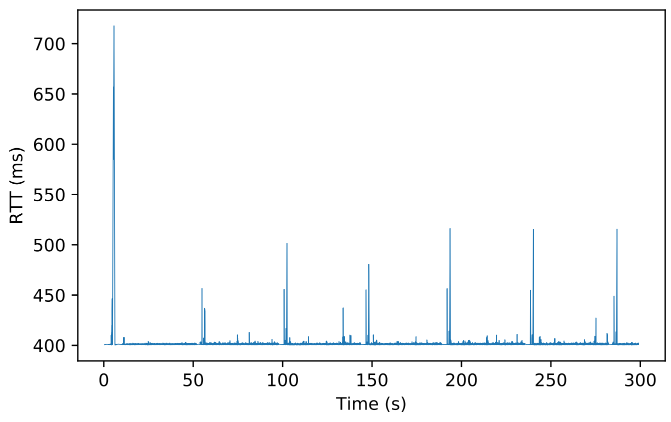


Fig. 5. Observed RTT LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 500 ms, with base RTT set to 400 ms.

So, the maximum window is determined by the reduction suffered after the exponential growth and how much LEDBAT++ can increase the window linearly during the limited time the linear increase phase lasts. LEDBAT++ achieves an equilibrium point, where these two effects compensate each other. We observe that in Fig. 4. After 200 s, LEDBAT++ enters in steady state where the growth during the linear increase phase matches exactly with the drop occurring right after the exit of the exponential growth. This additional penalty occurs when the RTT_b is larger than 300 ms (in the Annex we obtain an analytic expression that allows to compute that).

Takeaway: For rates of 20 Mps and base RTTs smaller than 20 ms, the LEDBAT++'s rate is close to the full capacity. For base RTTs between 20 ms and 300 ms, the periodic slow downs used in LEDBAT++ result in a 10% penalty on the rate achieved. For base RTTs larger than 300 ms, LEDBAT++ is unable to restore the rate after periodic slow downs, resulting in further penalty in the rate achieved.

5.1.2. LEDBAT++ solo performance in delay based mode: variation with capacity

We next look into how the rate achieved by LEDBAT++ varies with the link's capacity. We find that the range of RTT_b where we observe the different performance penalties is fairly insensitive to the capacity (at least for capacities between 1 and 40 Mbps) and that in all cases, for values of the RTT_b below 300 ms, the achieved rate is close to 0.9 of the available capacity while for RTT_b larger than that it drops further.

In Figs. 6–8 we plot the rate achieved by a single LEDBAT++ flow in a bottleneck link with a buffer of 500 ms varying the capacity with RTT set to 20 ms, 100 ms and 400 ms respectively. In both graphs we plot in addition the maximum capacity and 0.9 of the maximum capacity.

Takeaway: For bottleneck link capacities between 1 Mbps and 40 Mbps, we observe that for base RTTs lower than 300 ms LEDBAT++ is able to seize 90% of the available capacity while for larger base RTTs the throughput is further reduced. This is an effect of the periodic slow-downs implemented by LEDBAT++.

5.1.3. Possible solutions to the performance penalty problems

We have identified that the periodic slow downs used by LEDBAT++ result in two performance penalties, the $\frac{9}{10}$ penalty and the one resulting from the queuing delay experienced after the exponential growth. In order to address the $\frac{9}{10}$ penalty LEDBAT++ could restore the rate it was using before the periodic slow down without using the exponential increase. This would effectively limit the penalty to 2 RTTs significantly reducing the penalty. The downside with this approach is that after the 2 RTTs, the load of the bottleneck may have changed

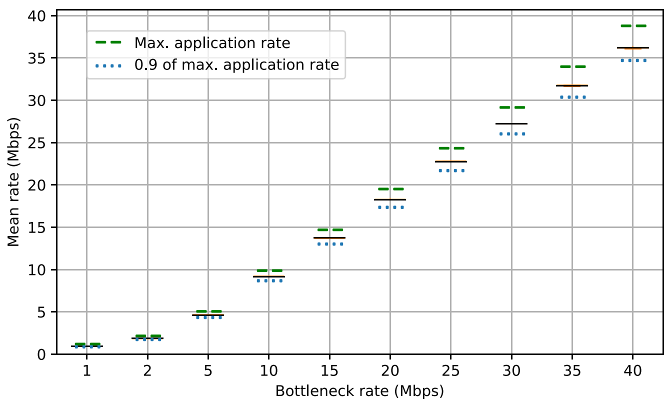


Fig. 6. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 500 ms, with a base RTT of 20 ms varying the capacity.

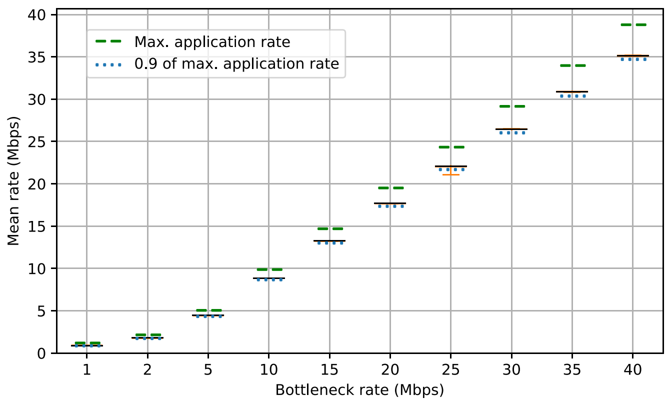


Fig. 7. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 500 ms, with an RTT of 100 ms varying the capacity.

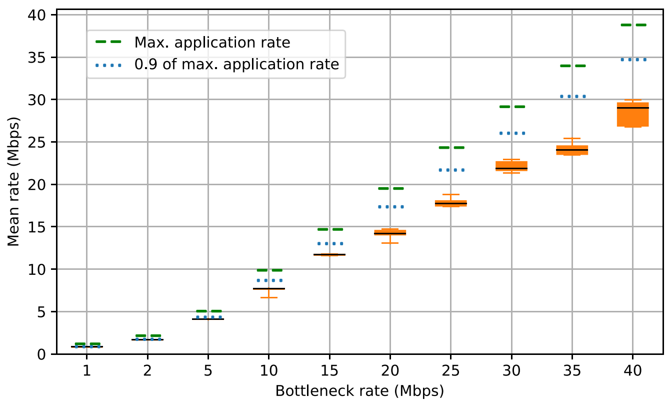


Fig. 8. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 500 ms, with an RTT of 400 ms varying the capacity.

and the restoration of the previous rate could result in some transient congestion. Other possible options to reduce this penalty could be to reduce the frequency of the periodic slow downs. For instance, BBR [13] performs slow downs every 10 s. Moreover, in its latest version 2, in order to reduce the penalty in performance, during slow-downs BBRv2 reduces its flightsize to half, which can also be a possible alternative to be considered for LEDBAT++.

Solutions to the penalty caused by the excess in queueing can be as simple as not considering the RTTs measured in the RTT after the exit of the exponential growth. Once the full window is restored, the ACK clocking ensures smooth operation without artificial added delays. The

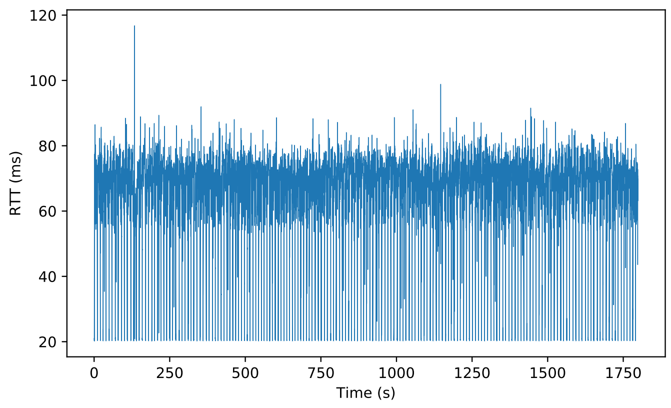


Fig. 9. RTT achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 500 ms, with an RTT of 20 ms varying the capacity.

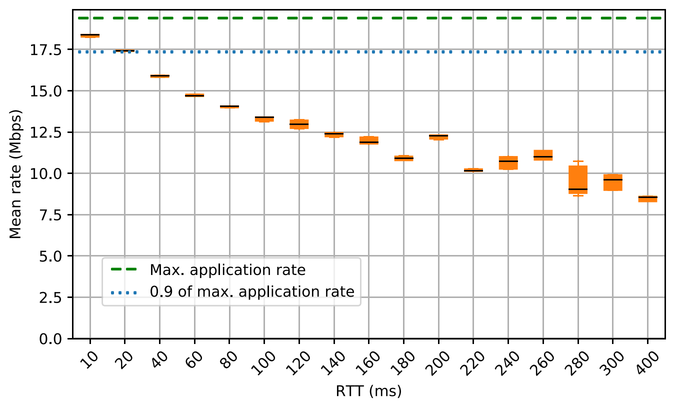


Fig. 10. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 30 ms, varying the base RTT.

potential downside of this approach is that if there is congestion at this moment, LEDBAT++ response is delayed one RTT.

In any case, performance of these simple solutions should be thoroughly analyzed and this is beyond the scope of this paper.

5.1.4. Latency drift

We also perform a set of long experiments to determine if LEDBAT++ manages to address the latency drift problem. We performed 8 runs of a single LEDBAT++ flow in a 20 Mbps bottleneck link with a 500 ms buffer and a 20 ms base RTT. Each run lasted for 30 mins. The RTT observed for one of these experiments is depicted in Fig. 9. We observe that because of periodic slow downs, LEDBAT++ is able to periodically measure the RTT correctly and that discarding earlier measurements does not cause the latency drift problem present in LEDBAT.

5.2. LEDBAT++ solo performance in loss based mode

For these experiments, we configure a small buffer in the bottleneck link. Specifically, we configure a buffer of size B equal to 30 ms.

5.2.1. LEDBAT++ solo performance in loss based mode: variation with RTT

As we did before, we analyze the variation with the RTT of the rate seized by a single LEDBAT++ flow when running on its own, but now with a small buffer in the bottleneck link.

We perform experiments computing the average rate achieved by a single LEDBAT++ flow running in a bottleneck link of capacity C equal to 20 Mbps, with a buffer of size B equal to 30 ms. We vary the base RTT (RTT_b) between 10 ms and 400 ms. Each experiment ran for 300 s and we computed the average rate. We perform 8 runs for each RTT measured. The results are plotted in Fig. 10. The graph plots the achieved rate at the application layer, and the maximum application layer rate possible in a 20 Mbps link is also plotted. In the graph we can observe that for RTT equal to 10 ms the rate achieved is close to the full capacity of the link, then with RTT equal to 20 ms the rate decreases to 0.9 of the available capacity and for RTTs larger than 40 ms the rates decrease even further.

In order to gain deeper insight of what is happening, we look at the evolution of the flight-size for base RTT equal to 20 ms in Fig. 11. We observe that the flight-size is most of the time larger than the BDP corresponding to the link (the BDP of capacity 20 Mbps and RTT of 20 ms is 47 kB[6]). The reason why LEDBAT++ is unable to seize all the available capacity in this case is because of the periodic slow downs. As in the case of large buffers, the periodic slow downs impose a 10% penalty in the capacity used, so LEDBAT++ in this case is able to use 90% of the capacity, as observed in Fig. 10.

[6] Using the correction factor associated to the application rate plotted in the graph.

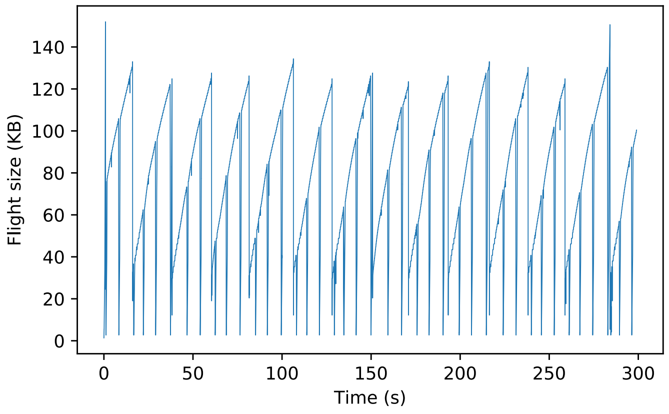


Fig. 11. Flight-size of a LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 30 ms, with base RTT set to 20 ms.

For smaller base RTTs, the penalty introduced by the periodic slow downs decreases, as the minimum window of 2 MSS used by LEDBAT++ during the slow downs is closer to the window required to achieve full utilization of the link.

We next look into larger base RTTs. In Fig. 12 we plot the flight-size for base RTT equal to 100 ms. We observe that the maximum flight-size observed matches to the BDP corresponding to the base RTT plus the available buffer (i.e., 310 kB), concluding that the losses occur because the buffer is full. However, because LEDBAT++ is functioning as a loss based AIMD, it is unable to reach the maximum capacity since the buffer is smaller than the BDP [14].

In Fig. 13 we plot the flight-size for base RTT equal to 260 ms. In this case, we observe that the maximum flight-size observed (610 kB) is considerably lower than the BDP corresponding to the base RTT plus the buffer (i.e., 767 kB). We also observe that LEDBAT++ fails to restore the window after a periodic slow down. So, in this case, we observe a similar behavior than the one exhibited in delay-based mode, that is that exponential growth used to restore the window after a periodic slow down fills up the buffer and cramps LEDBAT++ capacity to restore the window. In this case, the effect is more pronounced, since when the buffer is full, packets are lost (while in the delay based mode, larger delays are observed which trigger LEDBAT++'s delay based mechanisms).

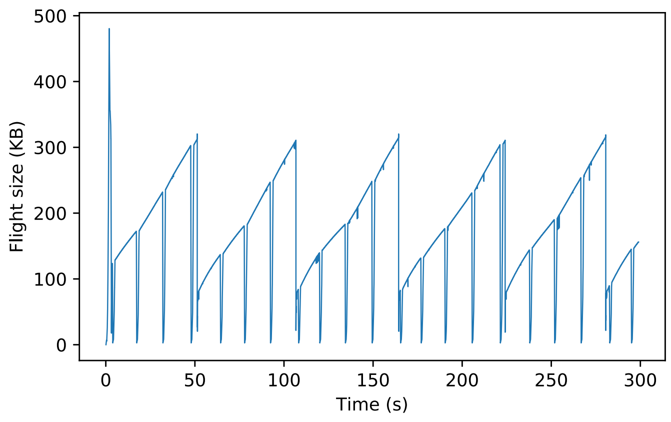


Fig. 12. Flight-size of a LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 30 ms, with base RTT set to 100 ms.

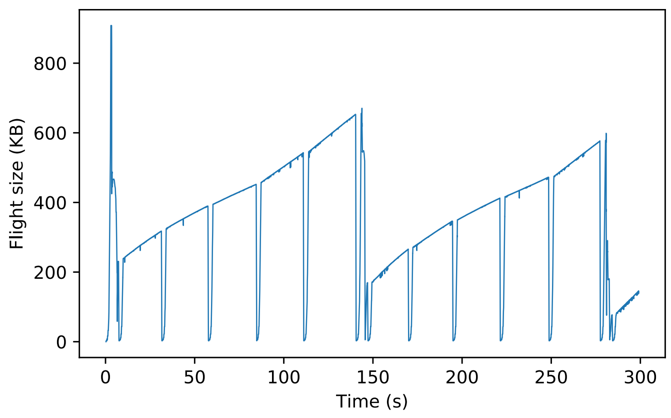


Fig. 13. Flight-size of a LEDBAT++ flow running solo in a bottleneck link of 20 Mbps with a buffer of 30 ms, with base RTT set to 260 ms.

Takeaway: In a bottleneck link of 20 Mbps, we find that for buffers of size B ms, smaller than the Target T, LEDBAT++ performance as a function of the base RTT is as follows. For base RTTs close to 2 ms, the full capacity is achieved, since the minimum window used by LEDBAT++ is enough to fill in the link. For base RTTs up to B ms, 0.9 of the capacity is achieved, since 10% of the capacity is lost due to inactivity periods caused by periodic slow downs. For larger base RTTs, LEDBAT++'s performance is penalized even further.

5.2.2. LEDBAT++ solo performance in loss based mode: variation with capacity

We next look into how the rate achieved by LEDBAT++ varies with the link's capacity. We find that the range of RTT_b where we observe the different behaviors are fairly insensitive to the capacity (at least for capacities between 1 and 40 Mbps) and that in all cases, for RTT_b equal to 20 ms, the achieved rate is close to 0.9 of the available capacity while for RTT_b equal to 100 ms it drops below that and for RTT_b equal to 300 ms it drops even further.

In Figs. 14–16 we plot the rate achieved by a single LEDBAT++ flow in a bottleneck link with a buffer of 30 ms varying the capacity with RTT_b set to 20 ms, 100 ms and 300 ms respectively. In both graphs we plot in addition the maximum capacity and 0.9 of the maximum capacity.

6. LEDBAT++ and Cubic

In this section, we look into how LEDBAT++ performs when sharing a bottleneck link with a TCP connection that is using Cubic as congestion control algorithm. We first consider the case of large bottleneck link buffers and then we move on to the case where small buffers are used.

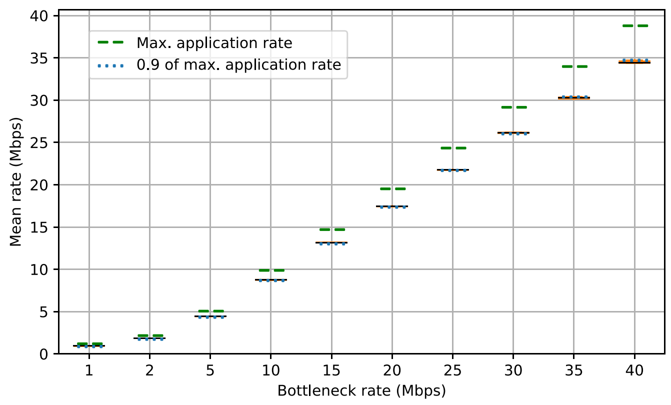


Fig. 14. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 30 ms, with a base RTT of 20 ms varying the capacity.

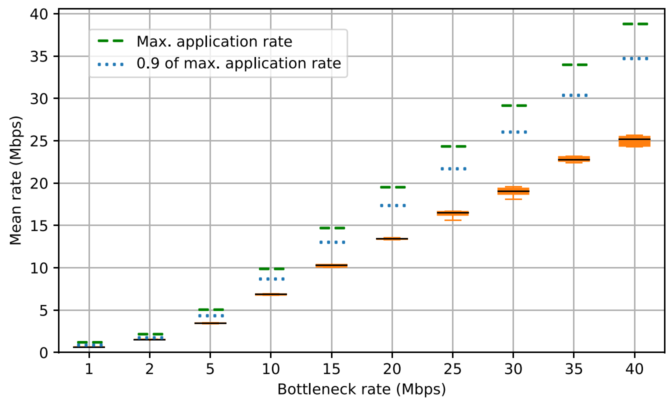


Fig. 15. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 30 ms, with an RTT of 100 ms varying the capacity.

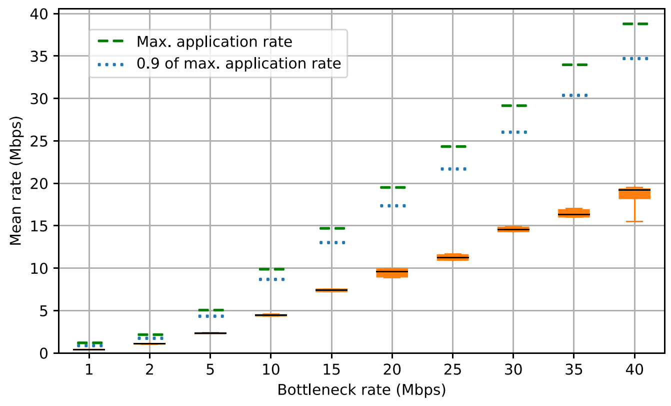


Fig. 16. Rate achieved by a single LEDBAT++ flow running solo in a bottleneck link with a buffer of 30 ms, with an RTT of 300 ms varying the capacity.

6.1. LEDBAT++ and Cubic with large buffers

In this case, we configure a large buffer on the bottleneck link of 500 ms. Because the buffer is much larger than the target T, LEDBAT++ should be able to react to excess of delays. In Fig. 17 we depict the rate achieved by a LEDBAT++ and a TCP/Cubic flow competing in a 20 Mbps bottleneck link for different values of the base RTT. For each

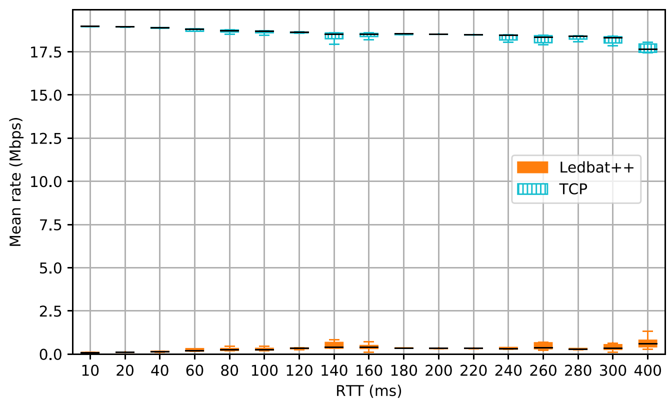


Fig. 17. Rate achieved by a single LEDBAT++ flow competing with a TCP/Cubic flow in a bottleneck link of 20 Mbps with a buffer of 500 ms, varying the base RTT.

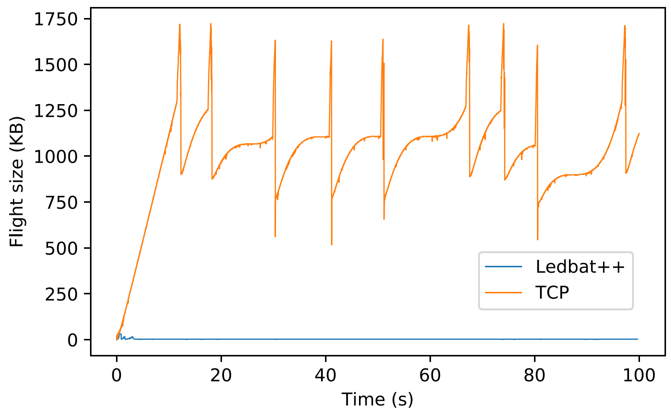


Fig. 18. Inflight packets for a single LEDBAT++ flow competing with a TCP/Cubic flow in a bottleneck link of with a buffer of 500 ms.

base RTT we performed 8 experiments. We observe that LEDBAT++ yields to Cubic, fulfilling its goal, irrespectively of the base RTT.[7]

In this case, the capacity seized by LEDBAT++ comes from two sources. First, LEDBAT++ always sends 2 MSS per RTT, in order to be able to continuously estimate the queuing delay and be able to ramp up when the observed delay is below the target T. The second source is the leftovers from Cubic, resulting from the reductions of the Cubic's congestion window due to losses. However, as it can be observed in the graph 18 that depicts the flight-size of both Cubic and LEDBAT++, despite of these effects, Cubic seizes most of the capacity.

We next measure the capacity split for bottleneck links of different capacities and present the results in Fig. 19. We perform 8 experiments for each capacity. Similarly than before, we observe that LEDBAT++ yields to Cubic for all measured capacities, achieving its goal.

6.1.1. Short TCP flows

We next investigate how LEDBAT++ background traffic impacts the completion time of short TCP flows. According to [16], 99.1% of flows carry less than 260 KB and less than 512 packets. In 20 we plot the completion time of a short TCP/Cubic flow when running alone in the link, when competing with a long TCP/Cubic flow and when competing with a long LEDBAT++ flow. We can observe that the completion time of the short flow is similar to the one that the flow obtains when running solo and significantly shorter than the one obtained when competing with another Cubic flow.

[7] We observe that for very large values of the base RTT, Cubic struggles a bit to reach the available capacity, as it has been previously observed in [15].

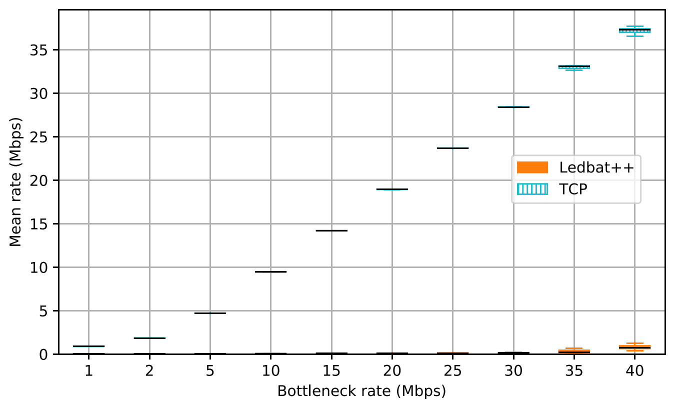


Fig. 19. Rate achieved by a single LEDBAT++ flow competing with a TCP/Cubic flow in a bottleneck link with a buffer of 500 ms, with 20 ms of base RTT and varying the capacity.

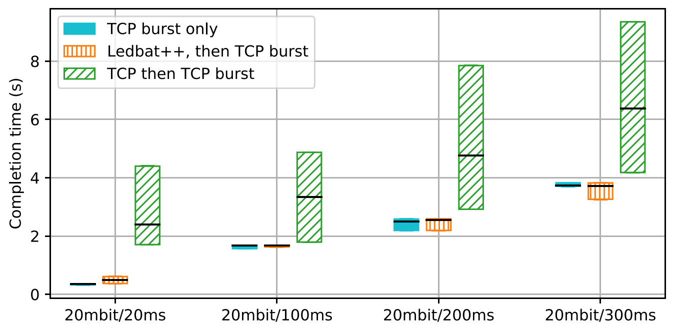


Fig. 20. Completion time of a short TCP/Cubic flow in 3 scenarios: running solo, competing with a long TCP/Cubic flow and competing with a long LEDBAT++ flow. The bottleneck link has 20 Mbps and a buffer of 500 ms, varying the base RTT.

6.2. LEDBAT++ and Cubic with small buffers

We now explore how LEDBAT++ behaves when sharing a bottleneck link with a Cubic flow if the buffer is smaller than the target. For that, we run experiments using a LEDBAT++ flow and a Cubic flow on a 20 Mbps bottleneck link with a 30 ms buffer and varying the base RTT. We run each experiment 8 times and the results are plotted in Fig. 21. We observe that LEDBAT++ uses less capacity than Cubic, but it does not yield completely, as in the case of larger buffers. That is expected since LEDBAT++'s delay-based mechanisms are not active and it only react to packet loss. But because of the GAIN parameter, LEDBAT manages to avoid competing in equal grounds than Cubic and yields part of the capacity to Cubic. Across the different values of the base RTT, Cubic uses at least 2.5 times more capacity than LEDBAT++ (in the case of the base RTT of 80 ms) and up to 10 times more capacity (for base RTT of 400 ms).

We also look into how LEDBAT++ competes with Cubic with small buffers for different capacities. We observe the results in Fig. 22 and we observe a similar behavior.

7. Inter-LEDBAT++ fairness

One of the main problems with the original LEDBAT are those related to inter-LEDBAT fairness and the so called late-comer advantage. We perform a number of experiments to learn if the proposed LEDBAT++ mechanisms achieve inter-LEDBAT++ fairness. We start with a number of experiments aimed to assess fairness between LEDBAT++ flows that start at the same time and then we perform other experiments to determine if the late-comer advantage still persists in LEDBAT++.

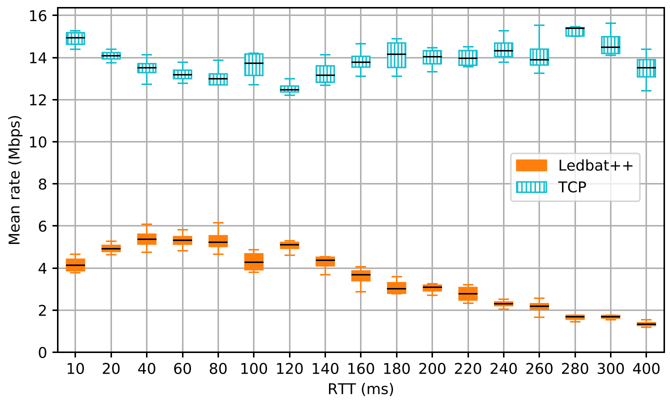


Fig. 21. Rate achieved by a single LEDBAT++ flow competing with a TCP/Cubic flow in a bottleneck link of 20 Mbps with a buffer of 30 ms, varying the base RTT.

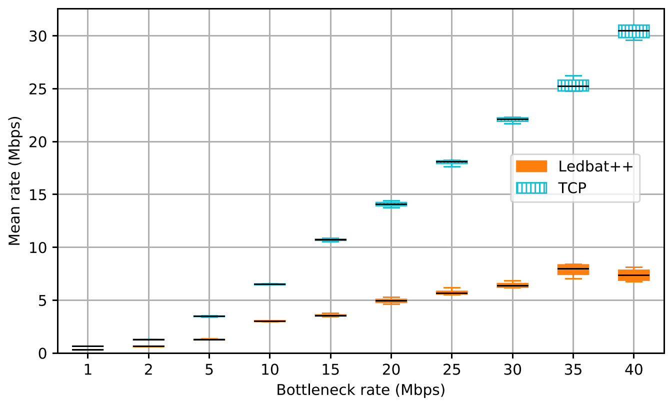
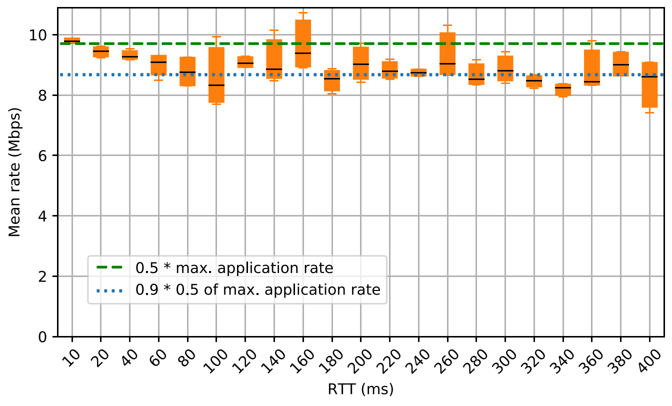
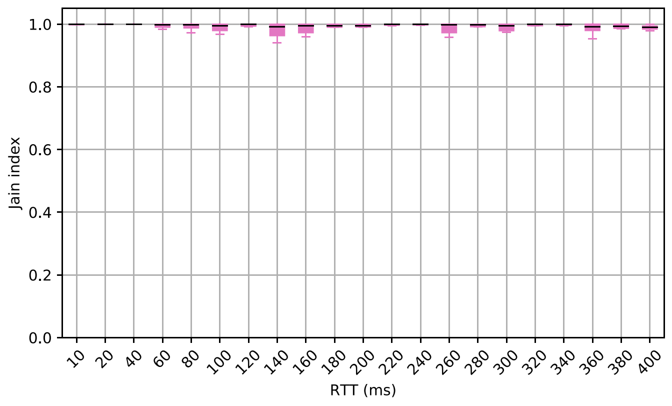


Fig. 22. Rate achieved by a single LEDBAT++ flow competing with a TCP/Cubic flow in a bottleneck link with a buffer of 30 ms, varying the capacity and using a base RTT of 20 ms.



(a) Rate achieved by a single LEDBAT++ flow.



(b) Jain's fairness index

Fig. 23. Fairness for two LEDBAT++ flows sharing a 20 Mbps bottleneck link, with both flows starting at the same time. The buffer is 500 ms and the base RTT varies.

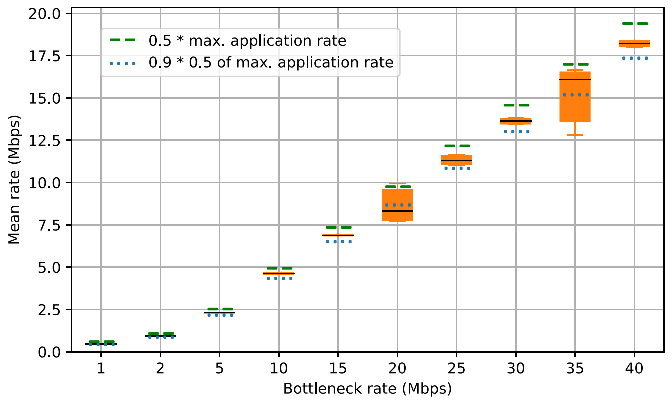


Fig. 24. Rate achieved by a single LEDBAT++ flow sharing a bottleneck link with another LEDBAT++ flow with both flows starting at the same time. The buffer is 500 ms, the base RTT is 100 ms and the capacity varies.

7.1. Fairness between synchronized LEDBAT++ flows

We measure the split achieved between two LEDBAT++ flows that start at the same and share a bottleneck link of 20 Mbps for different values of the base RTT. We perform 8 runs for each value of the base RTT and we plot the rate achieved by one of the flows in Fig. 23(a). We observe the flow never starves nor uses all available capacity. Most flows achieve a rate close to 45% of the available capacity (i.e., half of the 90% of the available capacity, that from previous experiments of a LEDBAT++ flow running solo, we know is the maximum LEDBAT++ can seize, accounting for the capacity wasted by the periodic slow downs). When looking at the deviations, we observe that the mean deviation (i.e., the difference between the rates achieves between the two flows) is generally around 15% and below 20%. We also plotted Jain's fairness index [17] for the two competing flows for the different RTTs in Fig. 23(b), and we find that it is always above 0.96 (1 being the ideal fair split).

We next perform similar experiments using bottleneck links with different capacities, to learn about the impact of the capacity on the fairness properties of LEDBAT++. In Fig. 24 we plot the capacity achieved by one of the LEDBAT flows for bottleneck links with different capacities and a base RTT of 100 ms. We observe that irrespectively of the capacity, a fair split is achieved.

7.2. Late-comer advantage

We next perform a set of experiments to assess whether LEDBAT++ suffers from the late-comer advantage present in LEDBAT. To understand the mechanics of LEDBAT++ in this scenario, we run an experiment with four LEDBAT++ flows that start with a difference of 10 s between each other. The flight-size for each of the flows is depicted in Fig. 25 and the RTT observed for each flow is plotted in Fig. 26.

We observe that when the first flow starts, it stabilizes around the rate required to meet the target queueing delay. The first flow was able to accurately measure the base RTT in the initial packets and also during the initial slow down. When the second flow starts, it is unable to properly measure the base RTT, as the first flow is bloating it up to reach the target queueing delay. So the second flow wrongfully

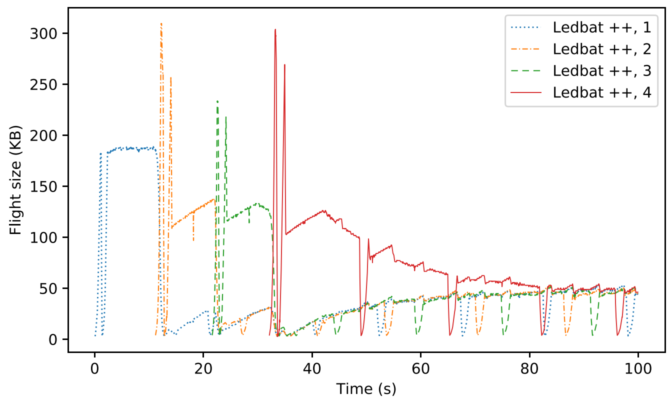


Fig. 25. Flight-size of four LEDBAT++ flows sharing a 20 Mbps bottleneck link starting in intervals of 10 s apart from each other. The buffer is 500 ms and the base RTT is 20 ms.

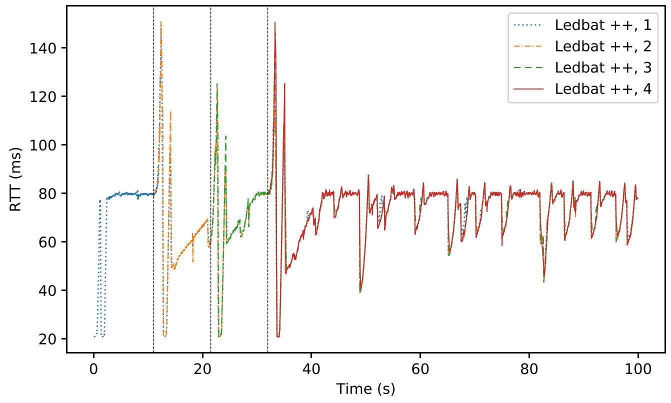


Fig. 26. RTT of four LEDBAT++ flows sharing a 20 Mbps bottleneck link starting in intervals of 10 s apart from each other. The buffer is 500 ms and the base RTT is 20 ms.

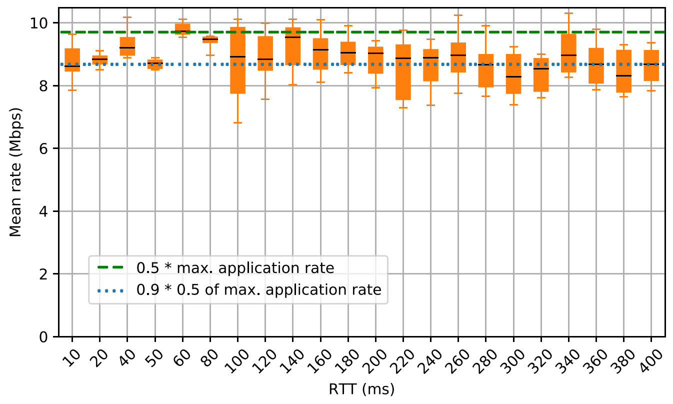


Fig. 27. Rate achieved by a single LEDBAT++ flow sharing a 20 Mbps bottleneck link with another LEDBAT++ flow that started 10 s later. The buffer is 500 ms and the base RTT varies.

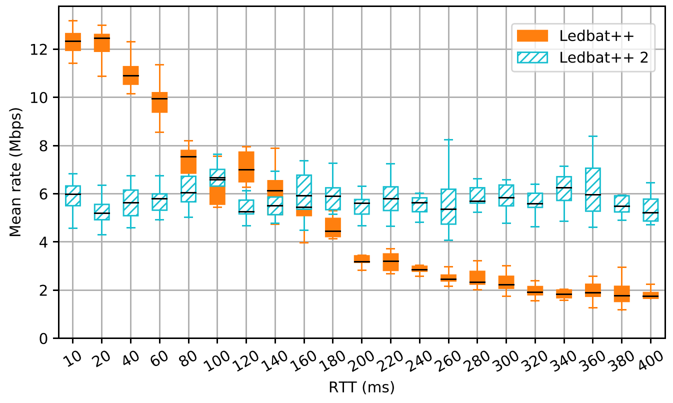


Fig. 28. Rate achieved by a LEDBAT++ flow with an RTT of 100 ms sharing a 20 Mbps bottleneck link with another LEDBAT++ flow with a different RTT. The buffer is 500 ms and the base RTT of the second flow varies. Both flows start at the same time.

measures a base RTT as the real base RTT plus the target. This means that it will then increase the sending rate until an additional queueing delay of T is achieved. The first flow, which has an accurate measure of the base RTT, now observes a queuing delay of twice the target and reduces its rate, yielding to the second flow. So far, this is the late-comer advantage effect that was present in LEDBAT.

However, once that the second flow has occupied the whole capacity available, LEDBAT++ exits slow-start and imposes an initial slow down. This can be observed in Fig. 25, i.e., the flight-size reduction of the second flow shortly after peaking. At this point, the buffer is fully emptied and the second flow is now able to properly measure the base RTT, as observed in Fig. 26. After the initial slowdown, flow 2 restores the previous congestion window, but now, with the proper assessment of the base RTT, it will reduce its sending rate towards a fair split with flow 1. The same mechanics occur when the third and fourth flow enters.

Once we understand the mechanics of how LEDBAT++ addresses the late-comer advantage problem, we present a series of experiments to measure the resulting capacity split between two LEDBAT++ flows that start at different times. In Fig. 27 we plot the rate of a LEDBAT++ flow sharing a bottleneck link of 20 Mbps with another LEDBAT++ flow. One of the flow starts 10 s after the other one. The rates plotted correspond to the flow that started first. We observe that the capacity is fairly split between them and there is no evidence of late comer advantage. We also computed Jain's fairness index for al the experiments and similarly to the case of flows starting at the same time, we find that for all the different RTTs, the index is above 0.94.

Takeaway: LEDBAT++ addresses the late comer advantage and converges to a fair split, albeit slowly.

7.3. RTT fairness

We next explore how the bottleneck link capacity is split between two LEDBAT++ flows with different RTTs. In Fig. 28 we plot the rates of two LEDBAT++ flows sharing a 20 Mbps link. One of the flows experiences a base RTT equal to 100 ms while the competing flow has an RTT that varies between 20 ms and 400 ms. In each experiment, both flows start at the same time.

We observe that LEDBAT++ lacks of RTT fairness and that flows with smaller base RTTs are able to seize more capacity than flows with larger RTTs. This is not surprising since LEDBAT++ is an ACK-clocked AIMD mechanism, as other RTT-unfair congestion control algorithms, which are known to allocate more capacity to flows with shorter RTTs [18].

8. Related work

To the best of our knowledge, there is no prior work measuring and analyzing LEDBAT++ performance.

There are a number of papers that analyzed and measured LEDBAT and proposed improvements to LEDBAT or simply alternative LBE congestion control algorithms.

Notably, in a series of papers [5,19–21], Rossi et al. analyze, simulate and measure different versions of the LEDBAT protocol and identified several of the shortcomings that LEDBAT++ aims to address as well as several solutions to the identified problems. Specifically, these papers identify the late-comer advantage problem in LEDBAT and evaluate possible approaches to address the issue. In particular, they propose and evaluate using a multiplicative decrease instead of additive increase as a possible fix to the problem. LEDBAT++ does incorporate the multiplicative decrease suggested in these papers, along with the periodic slow downs to overcome the late-comer advantage.

[22] studies the impact of route changes during the lifetime of a LEDBAT communication, which result in large delay variations. They conclude that such route changes would hinder performance and fairness of LEDBAT flows. While this paper does not suggests any solution to the identified problem, the final LEDBAT specification [1] did incorporate a maximum lifetime for the base delay measurements of 10 min, that attempts to address this concern. Unfortunately, the adopted solution causes in turn the latency drift problem instead. The latency drift problem is identified in [10]. They verify its existence through simulations. As a possible fix to the problem, the authors suggest stopping the transfer during a period at least equal to the target before updating the base delay measurements. This is essentially the approach taken by LEDBAT++, as it introduces periodic slow downs, that allows it to empty the buffer and measure the real base delay without self-interference.

[23] and more recently [24] characterize the performance of LEDBAT using different active queue management techniques, including PIE and CoDel.

Eclipse [25] is an alternative LBE congestion control algorithm that addresses some of the limitations of LEDBAT. As opposed to LEDBAT, that uses a fixed delay target, Eclipse uses a dynamic target that adapts to the network conditions. This is a different direction than the one followed by LEDBAT++ to address LEDBAT issues. To validate Eclipse, [25] performs a number of simulations using OMNeT++ in which they compare the performance of Eclipse with LEDBAT (not LEDBAT++ since Eclipse is prior to LEDBAT++) when running solo and also when competing with Reno.

In [26], the authors propose LEDBAT-MP, a multi-path capable congestion control algorithm based both on the original LEDBAT algorithm and on the SCTP multi-path transport. They evaluate the performance of the proposed LEDBAT-MP using a simulation setup using OMNeT++ and compare its performance against Reno and also against other multi-path capable congestion control algorithms. Like Eclipse, this work is prior to LEDBAT++ and as such, they do not consider it during the evaluation of LEDBAT-MP.

Meng et al. [27] propose a novel scavenger congestion control algorithm, PCC Proteus, that also aims to overcome the shortcomings identified for LEDBAT. Proteus is an extension to the PCC utility-based approach [28], that defines a new utility function to reflect the objectives of scavenger traffic. As part of the evaluation of Proteus, they perform a number of experiments involving LEDBAT, CUBIC, Proteus and a few other congestion control algorithms. Even though LEDBAT++ was already available in Windows 10 (through the Windows 10 anniversary update [7] and described in [27,29] tested the original LEDBAT algorithm instead of the newly proposed LEDBAT++. Consequently, they find the well know shortcomings of LEDBAT, notably, inter-LEDBAT fairness issues due to the late-comer advantage and competition with CUBIC on equal grounds when the buffer is not large enough to generate a queueing delay larger than LEDBAT's target. The conclusion is that Proteus outperforms LEDBAT, but no results are provided regarding to how LEDBAT++ performs against Proteus (nor any of the other congestion control algorithms analyzed).

9. Conclusions

We have experimentally studied the performance of LEDBAT++. We find that it roughly fulfills all the goals it was designed for. Notably, it fully yields in front of CUBIC when buffers larger than the Target are used in the bottleneck and it partially yields when small buffers are used. This is an improvement compared to LEDBAT, which competes in equal grounds with CUBIC for small buffers [6]. Also, LEDBAT++ addresses the late-comer advantage and the latency drift issues identified in LEDBAT, thanks to the periodic slow downs it performs.

However, we also identified that the very mechanism that allows LEDBAT++ to address these issues (i.e., the periodic slow down) imposes a performance penalty, impeding LEDBAT++ to fully seize all the available capacity when there is no competing traffic. We propose two simple solutions to suppress the penalty, namely, restore the rate without using slow start and disregard the delay samples during the RTT after the restoration of the rate.

As future work, we plan to implement the proposed solutions to the identified shortcomings in LEDBAT++ to validate their effectiveness. Also, we plan to extend the experimental study on the performance of LEDBAT++ to scenarios in which LEDBAT++ is competing with model-based congestion control algorithms, such as BBR [13], which is becoming increasingly popular.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by the EU EC through the NGI Pointer RIM project, Grant 871528, and the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M21), and in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

Appendix. Model for LEDBAT++ solo performance

A.1. $\frac{9}{10}^{th}$ Approximation for small and medium base RTTs

For small and medium base RTTs we can estimate the impact of the periodic slow downs as follows.

When not in a periodic slow down, the average congestion window (expressed in Bytes) is set to:

$$\overline{W} = C * (RTT_b + BUFF) \tag{5}$$

Periodically, LEDBAT++ reduces the window to 2 MSS for 2 RTTs and then uses slow-start to increase the window up to the value that it had before the periodic slow down. The behavior is repeated after 9 times the time that it took for slow-start to ramp back up the window.

We call T_{ss} the time that it takes for slow-start to increase the window from 2 MSS back to the value it had before the reduction. T_{ss} (expressed in RTTs) can be computed as:

$$T_{ss} = log_2(\frac{\overline{W}}{MSS}) \tag{6}$$

We can then express the observed behavior in terms of periodic slow down epochs, each one composed of: first, 2 RTTs during which the window is set to 2 MSS, second, a period of duration T_{ss} during which the window increase from 2 MSS to the average steady state window $\overline{W}$ using slow-start and third, a period of duration $9T_{ss}$ during which the window oscillates around $\overline{W}$. Hence the number of bytes sent in one slow down epoch can be calculated as:

$$B_{sent} = 2 * 2 * MSS + 2 * \overline{W} + 9 * T_{ss} * \overline{W} \tag{7}$$

the second term is the sum of segments sent during slow-start (geometric progression).

The maximum number of packets that could be sent during the same period can be approximated as:

$$B_{max} = 2 * C * RTT_b + 10 * T_{ss} * \overline{W} \tag{8}$$

This results in a ratio of the packets sent with respect to the maximum packets that can be sent using the full capacity of the bottleneck link equal to:

$$R = \frac{2 * 2 * MSS + 2 * \overline{W} + 9 * T_{ss} * \overline{W}}{2 * C * RTT_b + 10 * T_{ss} * \overline{W}} \tag{9}$$

For values of $\overline{W}$ that are large enough, which result in also a larger T_{ss}, the ratio of B_{sent} over B_{max} can be approximated by $\frac{9}{10}$.

A.2. Model for large base RTTs

For larger base RTTs, we have observed that the exponential growth results in a queueing delay larger than the target, forcing a reduction in the window immediately after exiting the exponential growth. After this reduction, LEDBAT++ observes a delay smaller than the target T and enters in linear increase, which is maintained until the next periodic slow down. The average rate is then determined by the magnitude of the initial decrease in the window right after existing the exponential growth and the overall duration of the linear increase period.

During exponential growth, each ACK received generates GAIN additional packets. If RTT_b is larger than 120 ms, GAIN is equal to 1. This means that, for base RTTs larger than 120 ms, if in the nth RTT k packets were sent, in the $(n+1)^{th}$ RTT, $2k$ packets will be issued. Because of the combination of exponential increase and ACK clocking, $2k$ packets will be issued during the time it takes the bottleneck to transmit k packets. So, packets will spend some (increasing) time in the buffer. If the queueing delay caused by these k additional packets in the last round of the exponential growth exceeds the target T, it triggers a subsequent window reduction.

According to the LEDBAT++ algorithm, the CW_{n+1} after measuring a queueing delay larger than the Target T in round n is computed as:

$$CW_{n+1} = \beta.CW_n \tag{10}$$

with

$$\beta = \frac{2.T - q_d}{T} \tag{11}$$

where q_d is the queuing delay experienced by the last packet of the last round of the exponential growth.[8]

We call $ssext$ the window size (measured in segments of size MSS) at which exponential growth is exited. When $ssext$ is approached using exponential growth, the last packet will experience a queuing delay equal to the transmission delay of the extra packets added during this round. If $ssext$ is a power of 2, $\frac{ssext}{2}$ packet are added. However, if $ssext$ is not a power of 2, the number of packets added in the last round is: $ssext - 2^{FLOOR(log_2(ssext))}$. This implies that:

$$\beta = \frac{2T - (ssext - 2^{FLOOR(log_2(ssext))}).\frac{MSS}{C}}{T} \tag{12}$$

since each added packet adds an extra $\frac{MSS}{C}$ delay.

The duration (in RTTs) of the exponential growth phase is $T_{ss} = log_2(ssext)$. The duration of the linear increase phase is then $9 * T_{ss}$. In steady state, the linear increase is equal to the decrease experienced right after exiting the exponential growth. This means that:

$$ssext(1 - \beta) = 9 * T_{ss} \tag{13}$$

Solving (numerically) Eq. (13) for $ssext$, we find that $ssext = 850KB$, which is close to the value observed in Fig. 8.

[8] LEDBAT++ applies a filter of the minimum of the last 4 packets, but we will neglect that here.

The queue generated after the exit of the exponential growth imposes a performance penalty when the window resulting from the reduction decreases below the BDP required to fill the capacity of the bottleneck link. Mathematically, this is expressed as:

$$\beta.ssext < C.RTT_b \tag{14}$$

Using the $ssext$ value computed earlier, we find this holds for values of RTT_b larger than 300 ms. This is aligned with what we observe in Fig. 2. We observe that for other RTTs larger than 300 ms, the $ssext$ is similar and independent of the RTT_b.

We perform similar computations for other values of the capacity, and we obtain similar RTT_b threshold values after which the penalty is experienced, i.e., the RTT_b threshold does not vary significantly with the bottleneck capacity. For instance, for a capacity of 40 Mbps, the RTT_b threshold computed is 330 ms.

References

[1] S. Shalunov, G. Hazel, J. Iyengar, M. Kuhlewind, Low extra delay background transport (LEDBAT), in: Request for Comments, (6817) RFC Editor, 2012.
[2] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger, CUBIC for Fast Long-Distance Networks, 8312, RFC Editor, 2018.
[3] S. Shalunovtold, This is how your BitTorrent downloads move so fast, 2013, Fast Company blog URL https://www.fastcompany.com/3014951/why-your-bittorrent-downloads-move-so-fast.
[4] LEDBAT Wikipedia entry, 2018, URL https://en.wikipedia.org/wiki/LEDBAT.
[5] G. Carofiglio, L. Muscariello, D. Rossi, S. Valenti, The quest for LEDBAT fairness, in: Global Communications Conference, 2010, GLOBECOM, 2010, pp. 1–6.
[6] P. Balasubramanian, O. Ertugay, D. Havey, LEDBAT++: Congestion Control for Background Traffic, draft-irtf-iccrg-ledbat-plus-plus-01, Internet Engineering Task Force, 2020, (in preparation) URL https://datatracker.ietf.org/doc/html/draft-irtf-iccrg-ledbat-plus-plus-01.
[7] P. Balasubramanian, LEDBAT++: LOw priority TCP congestion control in windows, 2017, Presentation in ICCRG meeting at IETF 100 URL https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-ledbat-low-priority-tcp-congestion-control-in-windows.
[8] A. Czechowski, et al., Fundamental concepts for content management in configuration manager, 2022, Microsoft Documentation URL https://docs.microsoft.com/en-us/mem/configmgr/core/plan-design/hierarchy/fundamental-concepts-for-content-management#windows-ledbat.
[9] D. Borman, R.T. Braden, Van Jacobson, R. Scheffenegger, TCP Extensions for high performance, in: Request for Comments, (7323) RFC Editor, 2014, http://dx.doi.org/10.17487/RFC7323, RFC 7323 URL https://rfc-editor.org/rfc/rfc7323.txt.
[10] D. Ros, M. Welzl, Assessing LEDBAT's delay impact, IEEE Commun. Lett. 17 (5) (2013) 1044–1047, http://dx.doi.org/10.1109/LCOMM.2013.040213.130137.
[11] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, Van Jacobson, BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time, Queue 14 (5) (2016) http://dx.doi.org/10.1145/3012426.3022184.
[12] E. Blanton, V. Paxson, M. Allman, TCP Congestion control, in: Request for Comments, (5681) RFC Editor, 2009, http://dx.doi.org/10.17487/RFC5681, RFC 5681 URL https://rfc-editor.org/rfc/rfc5681.txt.
[13] N. Cardwell, Y. Cheng, S.H. Yeganeh, I. Swett, Van Jacobson, BBR Congestion Control, draft-cardwell-iccrg-bbr-congestion-control-02, Internet Engineering Task Force, 2022, (in preparation) URL https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-02.
[14] N. Beheshti-Zavareh, Tiny Buffers for Electronic and Optical Routers (Ph.D. thesis), Stanford University, 2009.
[15] S. Ha, I. Rhee, L. Xu, CUBIC: A new TCP-friendly high-speed TCP variant, SIGOPS Oper. Syst. Rev. 42 (5) (2008) 64–74, http://dx.doi.org/10.1145/1400097.1400105.
[16] P. Jurkiewicz, G. Rzym, P. Boryło, Flow length and size distributions in campus internet traffic, 2018, arXiv e-prints arXiv:1809.03486.
[17] R.K. Jain, D.-M.W. Chiu, W.R. Hawe, et al., A Quantitative Measure of Fairness and Discrimination. Vol. 21, Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA, 1984.
[18] E. Gavaletz, J. Kaur, Decomposing RTT-unfairness in transport protocols, in: 2010 17th IEEE Workshop on Local Metropolitan Area Networks, LANMAN, 2010, pp. 1–6, http://dx.doi.org/10.1109/LANMAN.2010.5507159.
[19] D. Rossi, C. Testa, S. Valenti, L. Muscariello, LEDBAT: the new BitTorrent congestion control protocol, in: 2010 Proceedings of 19th International Conference on Computer Communications and Networks, IEEE, 2010, pp. 1–6.
[20] D. Rossi, C. Testa, S. Valenti, Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm, in: International Conference on Passive and Active Network Measurement, Springer, 2010, pp. 31–40.

[21] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, S. Valenti, Rethinking the low extra delay background transport (LEDBAT) protocol, Comput. Netw. 57 (8) (2013) 1838–1852.

[22] A.J. Abu, S. Gordon, Impact of delay variability on LEDBAT performance, in: 2011 IEEE International Conference on Advanced Information Networking and Applications, IEEE, 2011, pp. 708–715.

[23] Y. Gong, D. Rossi, C. Testa, S. Valenti, M.D. Taht, Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control, in: 2013 Proceedings IEEE INFOCOM, 2013, pp. 3291–3296, http://dx.doi.org/10.1109/INFCOM.2013.6567153.

[24] R. Al-Saadi, G. Armitage, J. But, Characterising LEDBAT performance through bottlenecks using PIE, FQ-CoDel and FQ-PIE active queue management, in: 2017 IEEE 42nd Conference on Local Computer Networks, LCN, IEEE, 2017, pp. 278–285.

[25] H. Adhari, T. Dreibholz, S. Werner, E.P. Rathgeb, Eclipse: A new dynamic delay-based congestion control algorithm for background traffic, in: 2015 18th International Conference on Network-Based Information Systems, 2015, pp. 115–123, http://dx.doi.org/10.1109/NBiS.2015.21.

[26] H. Adhari, S. Werner, T. Dreibholz, E. P. Rathgeb, LEDBAT-MP – on the application of "lower-than-best-effort" for concurrent multipath transfer, in: 2014 28th International Conference on Advanced Information Networking and Applications Workshops, 2014, pp. 765–771, http://dx.doi.org/10.1109/WAINA.2014.125.

[27] T. Meng, N.R. Schiff, P. Brighten Godfrey, M. Schapira, PCC proteus: Scavenger transport and beyond, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, in: SIGCOMM '20, Association for Computing Machinery, New York, NY, USA, 2020, http://dx.doi.org/10.1145/3387514.3405891.

[28] M. Dong, Q. Li, D. Zarchy, P. Brighten Godfrey, M. Schapira, PCC: Re-architecting congestion control for consistent high performance, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), USENIX Association, Oakland, CA, 2015, pp. 395–408, URL https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong.

[29] P. Balasubramanian, O. Ertugay, D. Havey, LEDBAT++: Congestion Control for Background Traffic, draft-balasubramanian-iccrg-ledbatplusplus-00, Internet Engineering Task Force, 2019, (in preparation) URL https://datatracker.ietf.org/doc/html/draft-balasubramanian-iccrg-ledbatplusplus-00.

Marcelo Bagnulo received the Electrical Engineering degree from the University of Uruguay and the Ph.D. degree in telecommunications from the Universidad Carlos III de Madrid (UC3M), Spain. Since 2008, he has been a tenured Associate Professor at UC3M. He has published more than 80 articles in the field of advanced communications in journals and congresses, including IEEE INFOCOM, ACM SIGCOMM, ACM Mobicom, ACM IMC, and IEEE/ACM TRANSACTIONS ON NETWORKING. He is the author of 21 RFCs in the Internet Engineering Task Force (IETF), including the Shim6 protocol for IPv6 multihoming and the NAT64/DNS64 tools suite for IPv6 transition. He has 26 H-index and 4258 total citations. His research interests include Internet architecture and protocols, interdomain routing, and security. From 2009 to 2011, he was a member of the Internet Architecture Board.

Alberto García-Martínez received the degree in telecommunication engineering, in 1995, and the Ph.D. degree in telecommunications, in 1999. In 1998, he joined the Universidad Carlos III de Madrid (UC3M), where he has been an Associate Professor, since 2001. He has published more than 50 articles in technical journals (IEEE/ACM Transactions on Networking, Computer Networks), magazines (IEEE Wireless Communications, IEEE Communications Magazine), and conferences. He has coauthored three RFCs. His main interests include interdomain routing, transport protocols, network security, and blockchain technologies.