Workgroup: Network Working Group
Internet-Draft:
draft-bagnulo-iccrg-iccrg-ledbat-bbr-
interop-00
Published: 8 February 2023
Intended Status: Informational
Expires: 12 August 2023
Authors: M. Bagnulo A. Garcia-Martinez
UC3M UC3M

LEDBAT++ BBR interoperability issues

Abstract

This document specifies describes some interoperability issues identified between LEDBAT++ and BBR, resulting in unexpected behaviour. Specifically, that under a set of common conditions, LEDBAT++ fails to yield in front of both BBRv1 and BBRv2(instead of the opposite expected behaviour).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 August 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in

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

Table of Contents

1. Introduction

BBR (Bottleneck Bandwidth and Round-trip propagation time) [I-D.cardwell-iccrg-bbr-congestion-control] is a model-based congestion control algorithm that attempts to improve the performance of Internet communications by reducing the delay (when bottleneck buffers are large) and increase the throughput (when bottleneck buffers are small). BBR estimates the queueing delay by monitoring the round-trip-time (RTT) and it detects congestion onset when the queue in the bottleneck's buffer starts to build up. BBR then adjusts its rate to send as fast as possible while avoiding the formation of queues in the bottleneck, operating closer to the optimal operation point. Two versions of BBR exist. BBRv1 is included in Linux since version 4.9, and it is currently widely used in the Internet. BBRv2 is being developed and it aims to overcome some limitations identified for BBRv1.

LEDBAT++ (Low Extra Delay Background Transport ++) [I-D.irtf-iccrg-ledbat-plus-plus] 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 Cubic or other loss-based congestion control algorithms, it reduces its sending rate earlier and more aggressively than competing flows, allowing Cubic traffic to use more of the available capacity. This effectively implements an LBE traffic class that has less priority than Cubic-TCP/best-effort traffic.

LEDBAT++ reacts both to packet loss and to variations in delay. Regarding packet loss, LEDBAT++ reacts with a multiplicative decrease, similar to most TCP congestion controllers. Regarding delay, LEDBAT++ aims for a target queueing delay T (i.e., on top of the base RTT), set to 60 ms.

Similarly to BBR, LEDBAT++ estimates the queueing delay by monitoring the RTT. When the measured queueing delay is below the target T, LEDBAT++ additively increases the sending rate and when the delay is above the target T, it multiplicatively reduces the sending rate. LEDBAT++ is the evolution of the original LEDBAT algorithm [RFC6817] that overcomes multiple limitations identified in the original specification.

Since BBR aims to provide a best effort service i.e. a traffic class with similar priority than Cubic and New Reno and LEDBAT++ is designed to provide a less-than-best-effort traffic class that yields in front of best effort traffic, it is expected that LEDBAT yields in front of BBR. On other other hand, BBR aims to operate closer to the point where there is no queue while LEDBAT++ is designed to operate with a queueing delay equal to thew target T, which may hint that LEDBAT++ is more aggressive than BBR (as it is willing to endure a longer queue). In the following sections, we report the results of some experiments that confirm that indeed in certain conditions, LEDBAT++ is more aggressive than BBR and because of that, LEDBAT++ fails to yield in front of BBR traffic, unfulfilling the design goals of both protocols. We detail the conditions when this happens and propose a possible solution to the identified interoperability problem.

2. BBR LEDBAT++ Interoperability issues

We performed a number of experiments to understand LEDBAT++ BBR interaction. We provide details about the experimental setup in section XX below. We found the following results:

Experiment 1: Two TCP connections with the same RTT, one of them using BBRv1 and the other one using LEDBAT++, compete for the capacity of the bottleneck link that is using a buffer that is large enough to generate a queueing delay of at least LEDBAT++'s target. We observe that:

- For base RTTs larger than T, LEDBAT++ behaves as a scavenger transport and yields in front of BBRv1.

- For base RTTs smaller than T, LEDBAT++ does not yield and seizes a significant share of the capacity (about half of the available capacity, depending on the specific RTT).

- Uncoordinated slowdowns of BBRv1 and LEDBAT++ flows are not enough to enable both flows to have visibility of the base RTT.

BBRv1 is unable to seize all the available capacity when the RTT is smaller than T because of its flightsize cap. Indeed, BBRv1 defines a flightsize cap of twice the bandwidth-delay product. This implies that the maximum queueing delay that a BBRv1 flow can generate is

one (additional) RTT. LEDBAT++ on the other hand, can tolerate a queuing delay of T. So, when the RTT is smaller than T, LEDBAT++ tolerates more queueing delay than BBRv1, which implies that the LEDBAT++ flow will bot back off when competing against a BBRv1 flow, explaining the observed behaviour.

Experiment 2: Two TCP connections with the same RTT, one of them using BBRv1 and the other one using LEDBAT++, compete for the capacity of the bottleneck link that is using a buffer that is capable of generating a maximum queueing delay of B seconds, B being smaller than LEDBAT++'s target T (i.e. same as Experiment 1 but with B smaller than T). We observe that:

*For base RTTs larger than B, LEDBAT++ behaves as a scavenger transport and yields in front of BBRv1.

*For base RTTs smaller than B, LEDBAT++ does not yield and seizes a significant share of the capacity (about half of the available capacity, depending on the specific RTT).

In this case, we observe a similar behaviour than in experiment 1, only that the tipping point when LEDBAT++ starts to yield in front of BBRv1 is determined by the buffer size. For RTTs smaller than the buffer size, the same justification used for the Experiment 1 results apply to explain why BBRv1 yields. For RTTs larger than B, then the BBRv1 flow is able to generate a queue that is alrge enough to create losses, which forces LEDBAT++ into packet loss mode, behaving less aggressively than Cubic and thus yielding in front of BBRv1

Experiment 3: Two TCP connections with the same RTT, one of them using BBRv2 and the other one using LEDBAT++, compete for the capacity of the bottleneck link that is using a buffer that is large enough to generate a queueing delay of at least LEDBAT++'s target. We observe that:

*For base RTTs larger than T, LEDBAT++ behaves as a scavenger transport and yields in front of BBRv2.

*For base RTTs smaller than T, BBRv2 yields in front of LEDBAT++, behaving exactly the opposite than expected.

Most modifications introduced in BBRv2 are intended to make it less aggressive, which is likely to explain the experiment's results. In particular, in BBRv2 the flightsize is not limited only by the flightsize cap, but also for the inflight model parameter that is aiming for a small queue, which seem to be at odds with LEDBAT++ actions to bloat the queue up to the target T. Also, BBRv2 aims to leave a headroom in the link to enable other flows to enter. It may

the case that this headroom is filled by LEDBAT++ traffic, enabling LEDBAT++ to seize more capacity.

Experiment 4: Two TCP connections with the same RTT, one of them using BBRv2 and the other one using LEDBAT++, compete for the capacity of the bottleneck link that is using a buffer that is capable of generating a maximum queueing delay of B seconds, B being smaller than LEDBAT++'s target T (i.e. same as Experiment 3 but with B smaller than T). We observe that:

*For base RTTs smaller than B, BBRv2 yields in front of LEDBAT++.

*For base RTTs larger than B, BBRv2 gradually seizes a larger share of the capacity, but it is not until much larger RTTs that is able to seize all the capacity.

BBRv2 incorporates new mechanisms to react to losses. When using buffers smaller that the target T, loss is the main congestion signal and both BBRv2 and LEDBAT++ respond to it ad the new BBRv2 mechanism seem to make it more aggressive, leaving some room for LEDBAT++ to seize.

3. Proposed solution

In our previous analysis, we posit that a BBR flow are unable to seize all the available capacity when competing against a LEDBAT++ flow due to its self-imposed limitation on the flightsize. This limitation is set to one additional BDP, and it is in place to limit the size of the resulting queue. When the RTT is smaller than the target T, the queue generated by the LEDBAT++ flow is larger than the one BBR is willing to generate, hence BBR yields.

In order to address this issue, we propose to limit the queue generated by LEDBAT++ to the minimum of the target T and the base RTT. This should solve the problem since this would result in LEDBAT++ refraining from generating queues larger than one BDP, so, it would consequently yield in from of BBR. We believe this would also positive for LEDBAT++ itself, as it would prevent it from bloating the RTT in relative terms compared to the base RTT for small base RTTs.

We have implemented the proposed solution and we have verified that it addresses the identified issue i.e. with the proposed solution, LEDBAT++ always yields in front of BBR.

4. Experiment data

In our experiments, we use the virtualised setup depicted in Figure 1, which features a dumbbell topology, that allows us to generate both LEDBAT++ and BBR flows that compete for the capacity of a

bottleneck link. We configure the characteristics of the bottleneck link, including the size of the buffer, the capacity and the delay. We use this topology for our experiments, because any path, no matter how complex it is, can be accurately modelled from the transport layer perspective 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.

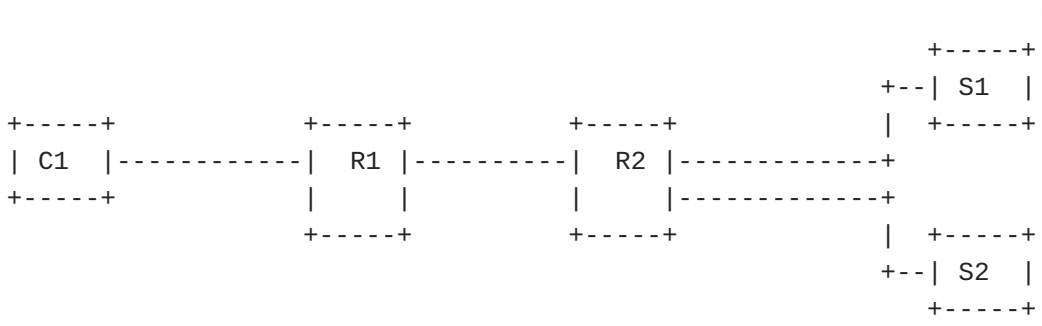

Figure 1: The rLEDBAT architecture.

C1, R1, R2 and S1 are Linux systems while S2 is a Windows 2019 Server with LEDBAT++ capability. S1 has BBR enabled. BBRv1 is already available in the Linux kernel and BBRv2 is installed. 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 tbf queueing discipline for the tc traffic control tool. A drop-tail buffer is configured in the R2-to-R1 link, with a size that we vary on different experiments, to represent different network setups. The links between S1 (S2) and R2 and the ones between C1 and R1 are configured with (much) larger capacities than the bottleneck link. During the experiments, we set the RTT of the path between S1 (S2) and C1 using tc netem.

In all the experiments, C1 connects to S1 and S2 nodes to perform downloads. Each flow is greedy, in the sense that it aims to transmit as much data as possible. Data is transferred using TCP, using BBR between C1 and S1 and LEDBAT++ between C1 and S2. TCP flow control never limits the communication rate, as we manually configure a large receiver window. To compute the rates for each flow, we start a tcpdump capture in C1. The MSS used is 1,390 bytes while the MTU is 1,456 bytes.

Additional data about the experiments, including graphs with the measurement results can be found at [COMNET]

5. Security Considerations

6. IANA Considerations

7. Acknowledgements

This work was supported by the EU through the StandICT CEL6 project.

8. Informative References

[COMNET] Bagnulo, M.B. and A.G. Garcia-Martinez, "When less is more: BBR versus LEDBAT++", , Computer Networks Volume 219, 2022.

[I-D.cardwell-iccrg-bbr-congestion-control] Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrg-bbr-congestion-control-02, 7 March 2022, <https://www.ietf.org/archive/id/draft-cardwell-iccrg-bbr-congestion-control-02.txt>.

[I-D.irtf-iccrg-ledbat-plus-plus] Balasubramanian, P., Ertugay, O., and D. Havey, "LEDBAT++: Congestion Control for Background Traffic", Work in Progress, Internet-Draft, draft-irtf-iccrg-ledbat-plus-plus-01, 25 August 2020, <https://www.ietf.org/archive/id/draft-irtf-iccrg-ledbat-plus-plus-01.txt>.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <https://www.rfc-editor.org/info/rfc6817>.

Authors' Addresses

Marcelo Bagnulo
UC3M

Email: marcelo@it.uc3m.es

Alberto Garcia-Martinez
UC3M

Email: alberto@it.uc3m.es