Network Working Group M. Bagnulo
Internet-Draft A. Garcia-Martinez
Intended status: Experimental UC3M
Expires: May 1, 2020 G. Montenegro
P. Balasubramanian
Microsoft
October 29, 2019

rLEDBAT: receiver-driven Low Extra Delay Background Transport for TCP
draft-bagnulo-iccrg-rledbat-01.txt

Abstract

This document specifies the rLEDBAT, a set of mechanisms that enable the execution of a less-than-best-effort congestion control algorithm for TCP at the receiver end.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

LEDBAT (Low Extra Delay Background Transport) [RFC6817] 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 [RFC8312] (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. 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 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 delay, LEDBAT aims for a target queueing delay. When the measured current queueing delay is below the target, LEDBAT increases the sending rate and when the delay is above the target, it reduces the sending rate. LEDBAT estimates the queuing delay by subtracting the measured current one-way delay from the estimated base one-way delay (i.e. the one-way delay in the absence of queues).

The LEDBAT specification [RFC6817] defines the LEDBAT congestion-control algorithm, implemented in the sender to control its sending rate. LEDBAT is specified in a protocol and layer agnostic manner.

LEDBAT++ [I-D.balasubramanian-iccrg-ledbatplusplus] is also an LBE congestion control algorithm which is inspired in LEDBAT while addressing several problems identified with the original LEDBAT specification. In particular the differences between LEDBAT and LEDBAT++ include: i) LEDBAT++ uses the round-trip-time (RTT) (as opposed to the one way delay used in LEDBAT) to estimate the queuing delay; ii) LEDBAT++ uses an Additive Increase/Multiplicative Decrease algorithm to achieve inter-LEDBAT++ fairness and avoid the late-comer advantage observed in LEDBAT; iii) LEDBAT++ performs periodic slowdowns to improve the measurement of the base delay; iv) LEDBAT++ is defined for TCP.

In this note, we describe rLEDBAT, a set of mechanisms that enable the execution of an LBE delay-based congestion control algorithm such as LEDBAT++ in the receiver end of a TCP connection.

2. Motivations for rLEDBAT

rLEDBAT enables new use cases and new deployment models, fostering the use of LBE traffic and benefitting the global Internet by improving overall allocation of resources. The following scenarios are enabled by rLEDBAT:

Content Delivery Networks and more sophisticated file distribution scenarios: Consider the case where the source of a file to be distributed (e.g., a software developer that wishes to distribute a software update) would prefer to use LBE and it enables LEDBAT/LEDBAT++ in the servers containing the source file. However, because the file is being distributed through a CDN which surrogates do not support LBE congestion control, the result is that the file transfers, originated from CDN surrogates will not be using LBE. Interestingly enough, in the case of the software update, the developer may also control the software performing the download in the client, the receiver of the file, but because current LEDBAT/LEDBAT++ are sender-based algorithms, controlling the client is not enough to enable LBE congestion control in the communication. rLEDBAT would enable the use of LBE traffic class for file distribution in this setup.

Interference from proxies and other middleboxes: Proxies and other middleboxes are a commonplace in the Internet. For instance, in the case of mobile networks, proxies are frequently used. In the case of enterprise networks, it is common to deploy corporate proxies for filtering and firewalling. In the case of satellite

links, Performance Enhancement Proxies (PEPs) are deployed to mitigate the effect of the long delay in TCP connection. These proxies terminate the TCP connection on both ends and prevent the use of LBE congestion control in the segment between the proxy and the sink of the content, the client. By enabling rLEDBAT, clients would be able to enable LBE traffic between them and the proxy.

Receiver-defined preferences. It is frequent that the bottleneck of the communication is the access link. This is particularly true in the case of mobile devices. It is then especially relevant for mobile devices to properly manage the capacity of the access link. With current technologies, it is possible for the mobile device to use different congestion control algorithms expressing different preferences for the traffic. For instance, a device can choose to use standard-TCP for some traffic and to use LEDBAT/LEDBAT++ for other traffic. However, this would only affect the outgoing traffic since both standard-TCP and LEDBAT/LEDBAT++ are sender-driven. The mobile device has no means to manage the traffic in the down-link, which is in most cases, the communication bottleneck for a typical eye-ball end-user. rLEDBAT enables the mobile device to selectively use LBE traffic class for some of the incoming traffic. For instance, by using rLEDBAT, a user can use regular standard-TCP/UDP for video stream (e.g., Youtube) and use rLEDBAT for other background file download.

3. rLEDBAT mechanisms

rLEDBAT provides the mechanisms to implement an LBE congestion control algorithm at the receiver-end of a TCP connection. The rLEDBAT receiver controls the sender's rate through the Receive Window announced to the sender in the TCP header.

rLEDBAT assumes that the sender is a standard TCP sender. rLEDBAT does not require any rLEDBAT-specific modifications to the TCP sender. The envisioned deployment model for rLEDBAT is that the clients implement rLEDBAT and this enables rLEDBAT in communications with existent standard TCP senders. In particular, the sender MUST implement [I-D.ietf-tcpm-rfc793bis] and it also MUST implement the Time Stamp Option as defined in [RFC7323]. Also, the sender SHOULD implement some of the standard congestion control mechanisms, such as Cubic [RFC8312] or New Reno [RFC5681].

rLEDBAT does not defines a new congestion control algorithm. The LBE congestion control algorithm executed in the rLEDBAT receiver is defined in other documents. The rLEDBAT receiver MUST use an LBE congestion control algorithm. Because rLEDBAT assumes a standard TCP sender, the sender will be using a "best effort" congestion control algorithm (such as Cubic or New Reno). Since rLEDBAT uses the

Receive Window to control the sender's rate and the sender calculates the sender's window as the minimum of the Receive window and the congestion window, rLEDBAT will only be effective as long as the congestion control algorithm executed in the receiver yields a smaller window than the one calculated by the sender. This is normally the case when the receiver is using an LBE congestion control algorithm. The rLEDBAT receiver SHOULD use the LEDBAT++ congestion control algorithm [I-D.balasubramanian-iccrg-ledbatplusplus]. The rLEDBAT MAY use other LBE congestion control algorithms defined elsewhere as long as they use the round-trip-time as input to estimate the queueing delay as rLEDBAT as currently defined does not provide the means for the sender to estimate the queueing delay using one way delay measurements.

EDITOR'S NOTE: should we recommend the use of LEDBAT for rLEDBAT as long as the RTT is used to estimate the queueing delay?

Irrespectively of which congestion control algorithm is executed in the receiver, an rLEDBAT connection will never be more aggressive than standard TCP since it is always bounded by the congestion control algorithm executed at the sender.

rLEDBAT is essentially composed of three types of mechanisms, namely, those that provide the means to measure the round trip time, mechanisms to detect packet loss and the means to manipulate the Receive Window to control the sender's rate. We describe them next.

3.1. Controlling the receive window

rLEDBAT uses the Receive Window (RCV.WND) of TCP to enable the receiver to control the sender's rate. [I-D.ietf-tcpm-rfc793bis] defines that the RCV.WND is used to announce the available receive buffer to the sender for flow control purposes. In order to avoid confusion, we will call fc.WND the value that a standard RFC793bis TCP receiver calculates to set in the receive window for flow control purposes. We call rl.WND the window value calculated by rLEDBAT algorithm and we call RCV.WND the value actually included in the Receive Window field of the TCP header. For a RFC793bis receiver, RCV.WND == fc.WND.

In the case of rLEDBAT receiver, the rLEDBAT receiver MUST NOT set the RCV.WND to a value larger than fc.WND and it SHOULD set the RCV.WND to the minimum of rl.WND and fc.WND, honoring both.

When using rLEDBAT, two congestion controllers are in action in the flow of data from the sender to the receiver, namely, the congestion control algorithm of TCP in the sender side and the LBE congestion

control algorithm executed in the receiver and conveyed to the sender through the RCV.WND. In the normal TCP operation, the sender uses the minimum of the congestion window cwnd and the receiver window RCV.WND to calculate the sender's window SND.WND. This is also true for rLEDBAT, as the sender is a regular TCP sender. This guarantees that the rLEDBAT flow will never transmit more aggressively than a TCP flow, as the sender's congestion window limits the sending rate. Moreover, because a LBE congestion control algorithm such as LEDBAT/LEDBAT++ is designed to react earlier and more aggressively to congestion than regular TCP congestion control, the rl.WND contained in the RCV.WND field of TCP will be in general smaller than the congestion window calculated by the TCP sender, implying that the rLEDBAT congestion control algorithm will be effectively controlling the sender's window.

In summary, the sender's window is: SND.WND = min(cwnd, rl.WND, fc.WND)

3.1.1. Avoiding window shrinking

The LEDBAT/LEDBAT++ algorithm executed in a rLEDBAT receiver increases or decreases the rl.WND according to congestion signals (variations on the estimations of the queueing delay and packet loss). If the new value for rl.WND is smaller than the current one then directly announcing it in the RCV.WND may result in shrinking the window, i.e., moving the right window edge to the left. Shrinking the window is discouraged as per [I-D.ietf-tcpm-rfc793bis], as it may cause unnecessary packet loss and performance penalty. To be consistent with [I-D.ietf-tcpm-rfc793bis], the rLEDBAT receiver SHOULD NOT shrink the receive window.

In order to avoid window shrinking, upon the reception of a data packet, the announced window can be reduced in the number of bytes contained in the packet at most. This may fall short to honor the new calculated value of the rl.WND. So, in order to reduce the window as dictated by the rLEDBAT algorithm, the receiver SHOULD progressively reduce the advertised RCV.WND, always honoring that the reduction is less or equal than the received bytes, until the target window determined by the rLEDBAT algorithm is reached. This implies that it may take up to one RTT for the rLEDBAT receiver to drain enough in-flight bytes to completely close its receive window without shrinking it. This is more than sufficient to honor the window output from the LEDBAT/LEDBAT++ algorithms since they only allows to perform at most one multiplicative decrease per RTT.

3.1.2. Window Scale Option

The Window Scale (WS) option [RFC7323] is a mean to increase the maximum window size permitted by the Receive Window. The use of the WS option implies that the changes in the window are expressed in the units resulting of the WS option used in the TCP connection. This means that the rLEDBAT client will have to accumulate the increases resulting from the different received packets, and only convey a change in the window when the accumulated sum of increases is equal or higher than one unit used to express the receive window according to the WS option in place for the TCP connection.

Changes in the receive window that are smaller than 1 MSS are unlikely to have any immediate impact on the sender's rate, as usual TCP segmentation practice results in sending full segments (i.e., segments of size equal to the MSS). So, accumulating changes in the receive window until completing a full MSS in the sender or in the receiver makes little difference.

Current WS option specification [RFC7323] defines that allowed values for the WS option are between 0 and 14. Assuming a MSS around 1500 bytes, WS option values between 0 and 11 result in the receive window being expressed in units that are about 1 MSS or smaller. So, WS option values between 0 and 11 have no impact in rLEDBAT.

WS option values higher than 11 can affect the dynamics of rLEDBAT, since control may become too coarse (e.g., with WS of 14, a change in one unit of the receive window implies a change of 10 MSS in the effective window).

For the above reasons, the rLEDBAT client SHOULD set WS option values lower than 12. Additional experimentation is required to explore the impact of larger WS values in rLEDBAT dynamics.

Note that the recommendation for rLEDBAT to set the WS option value to lower values does not precludes the communication with servers that set the WS option values to larger values, since the WS option value used is set independently for each direction of the TCP connection.

3.2. Measuring the Round Trip Time

LEDBAT++ measures base and current RTT to estimate the queueing delay. In the next sections we describe how rLEDBAT mechanisms enable the receiver to measure the RTT.

The original LEDBAT algorithm uses the one-way delay to estimate the queuing delay. We have encountered a number of issues when

attempting to measure the one-way delay in TCP, which resulted in deferring the recommendation of the use of one-way delay to estimate the queuing delay in rLEDBAT for the future, when additional research is done in this space. We describe the difficulties encountered in the Appendix below.

3.2.1. Measuring the RTT to estimate the queueing delay

LEDBAT++ uses the round trip time (RTT) to estimate the queueing delay. In order to estimate the queueing delay using the RTT, the rLEDBAT receiver estimates the base RTT (i.e., the constant components of the RTT) and also measures the current RTT. By subtracting these two values, we obtain the queuing delay to be used by the rLEDBAT controller.

LEDBAT++ discovers the base RTT (RTTb) by taking the minimum value of the measured RTTs over a period of time. The current RTT (RTTc) is estimated using a number of recent samples and applying a filter, such as the minimum (or the mean) of the last k samples. Using the RTT to estimate the queueing delay has a number of shortcomings and difficulties that we discuss next.

The queuing delay measured using the RTT includes also the queueing delay experienced by the return packets in the direction from the rLEDBAT receiver to the sender. This is a fundamental limitation of this approach. The impact of this error is that the rLEDBAT controller will also react to congestion in the reverse path direction which results in an even more conservative mechanism.

In order to measure the RTT, the rLEDBAT client MUST enable the Time Stamp (TS) option [RFC7323]. By matching the TSVal value carried in outgoing packets with the TSecr value observed in incoming packets, it is possible to measure the RTT. This allows the rLEDBAT receiver to measure the RTT even if it is acting as a pure receiver. In a pure receiver there is no data flowing from the rLEDBAT receiver to the sender, making impossible to match data packets with acknowledgements packets to measure the RTT, as it is usually done in TCP for other purposes.

Depending on the frequency of the local clock used to generate the values included in the TS option, several packets may carry the same TSVal value. If that happens, the rLEDBAT receiver will be unable to match the different outgoing packets carrying the same TSVal value with the different incoming packets carrying also the same TSecr value. However, it is not necessary for rLEDBAT to use all packets to estimate the RTT and sampling a subset of in-flight packets per RTT is enough to properly assess the queueing delay. The RTT MUST then be calculated as the time since the first packet with a given

TSVal was sent and the first packet that was received with the same value contained in the TSecr. Other packets with repeated TS values SHOULD NOT be used for the RTT calculation.

Several issues must be addressed in order to avoid an artificial increase of the observed RTT. Different issues emerge depending whether the rLEDBAT capable host is sending data packets or pure ACKs to measure the RTT. We next consider the issues separately.

3.2.1.1. Measuring RTT sending pure ACKs

In this scenario, the rLEDBAT node (node A) sends a pure ACK to the other endpoint of the TCP connection (node B), including the TS option. Upon the reception of the TS Option, host B will copy the value of the TSVal into the TSecr field of the TS option and include that option into the next data packet towards host A. However, there are two reasons why B may not send a packet immediately back to A, artificially increasing the measured RTT. The first reason is when A has no data to send. The second is when A has no available window to put more packets in-flight. We describe next how each of these cases is addressed.

The case where the host B has no data to send when it receives the pure Acknowledgement is expected to be rare in the rLEDBAT use cases. rLEDBAT will be used mostly for background file transfers so the expected common case is that the sender will have data to send throughout the lifetime of the communication. However, if, for example, the file is structured in blocks of data, it may be the case that seldom, the sender will have to wait until the next block is available to proceed with the data transfer and momentarily lack of data to send. To address this situation, the filter used by the congestion control algorithm executed in the receiver SHOULD discard the larger samples (e.g. a min filter would achieve this) when measuring the RTT using pure ACK packets.

The limitation of available sender's window to send more packets can come either from the TCP congestion window in host B or from the announced receive window from the rLEDBAT in host A. Normally, the receive window will be the one to limit the sender's transmission rate, since the LBE congestion control algorithm used by the rLEDBAT node is designed to be more restrictive on the sender's rate than standard-TCP. If the limiting factor is the congestion window in the sender, it is less relevant if rLEDBAT further reduces the receive window due to a bloated RTT measurement, since the rLEDBAT is not actively controlling the sender's rate. Nevertheless, the proposed approach to discard larger samples would also address this issue.

To address the case in which the limiting factor is the receive window announced by rLEDBAT, the congestion control algorithm at the receiver SHOULD discard the RTT measurements done using pure ACK packets while reducing the window and avoid including bloated samples in the queueing delay estimation. The rLEDBAT receiver is aware whether a given TSVal value was sent in a pure ACK packet where the window was reduced, and if so, it can discard the corresponding RTT measurement.

3.2.1.2. Measuring the RTT sending data packets

In the case that the rLEDBAT node is sending data packets and matching them with pure ACKs to measure the RTT, a factor that can artificially increase the RTT measured is the presence of delayed Acknowledgements. According to the TS option generation rules [RFC7323], the value included in the TSecr for a delayed ACK is the one in the TSVal field of the earliest unacknowledged segment. This may artificially increase the measured RTT.

If both endpoints of the connection are sending data packets, Acknowledgments are piggybacked into the data packets and they are not delayed. Delayed ACKs only increase the RTT measurement in the case that the sender has no data to send. Since the expected use case for rLEDBAT is that the sender will be sending background traffic to the rLEDBAT receiver, the cases where delayed ACKs increase the measured RTT are expected to be rare.

Nevertheless, for those measurements done using data packets sent by the rLEDBAT node matching pure ACKs sent from the other endpoint of the connection, they will result in an increased RTT. The additional increase in the measured RTT will range between the transmission delay of on packet and 500 ms. The reason for this is that delayed ACKs are generated every second data packet received and not delayed more than 500 ms according to [I-D.ietf-tcpm-rfc793bis]. The rLEDBAT receiver MAY discard the RTT measurements done using data packets from the rLEBDAT receiver and matching pure ACKs, especially if it has recent measurements done using other packet combinations. Also, applying a filter that discard larger samples would also address this issue (e.g. a min filter).

3.3. Detecting retransmissions and packet losses

The rLEDBAT receiver is capable of detecting retransmitted packets in the following way. We call RCV.HGH the highest sequence number correspondent to a received byte of data (not assuming that all bytes with smaller sequence numbers have been received already, there may be holes) and we call TSV.HGH the TSVal value corresponding to the segment in which that byte was carried. SEG.SEQ stands for the

sequence number of a newly received segment and we call TSV.SEQ the TSVal value of the newly received segment.

If SEG.SEQ < RCV.HGH and TSV.SEQ > TSV.HGH then the newly received segment is a retransmission. This is so because the newly received segment was generated later than another already received segment which contained data with a larger sequence number. This means that this segment was lost and was retransmitted.

The proposed mechanism to detect retransmissions at the receiver fails when there are window tail drops. If all packets in the tail of the window are lost, the receiver will not be able to detect a mismatch between the sequence numbers of the packets and the order of the timestamps. In this case, rLEDBAT will not react to losses but the TCP congestion controller at the sender will, most likely, reduce its window to 1MSS and take over the control of the sending rate, until slow start ramps up and catches the current value of the rLEDBAT window.

4. Security Considerations

5. IANA Considerations

6. Acknowledgements

This work was supported by the EU through the H2020 5G-RANGE project and by the Spanish Ministry of Economy and Competitiveness through the 5G-City project (TEC2016-76795-C6-3-R).

7. Informative References

[I-D.balasubramanian-iccrg-ledbatplusplus]
Balasubramanian, P., Ertugay, O., and D. Havey, "LEDBAT++: Congestion Control for Background Traffic", draft-balasubramanian-iccrg-ledbatplusplus-00 (work in progress), July 2019.

[I-D.ietf-tcpm-rfc793bis]
Eddy, W., "Transmission Control Protocol Specification", draft-ietf-tcpm-rfc793bis-14 (work in progress), July 2019.

[khono] Kohno, T., Broido, A., and K. Claffy, "Remote physical device fingerprinting", IEEE Transactions on Dependable and Secure Computing Vol 2, Number 2, 2005.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[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>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

Appendix A. **Difficulties while measuring one way delay at the TCP** receiver

The LEDBAT algorithm uses the one-way delay of packets as input. A TCP receiver can measure the delay of incoming packets directly (as opposed to the sender-based LEDBAT, where the receiver measures the one-way delay and needs to convey it to the sender).

In the case of TCP, the receiver can use the Time Stamp option to measure the one way delay by subtracting the time stamp contained in the incoming packet from the local time at which the packet has arrived.

In order to measure the one way delay using TCP timestamps, the rLEDBAT receiver needs to discover the units in which the values of the TS option are expressed and second, to account for the skew between the two clocks of the endpoints of the TCP connection. Note that a mismatch of 100 ppm (parts per million) in the estimation at the receiver of the clock rate of the sender accounts for 6 ms of variation per minute in the measured delay for a communication, just one order of magnitude below the target set for controlling the rate by rLEDBAT. Typical skew for untrained clocks is reported to be around 100-200 ppm [RFC6817].

In order to learn both the TS units and the clock skew, the rLEDBAT receiver compares how much local time has elapsed between the sender has issued two packets with different TS values. By comparing the local time difference and the TS value difference, the receiver can assess the TS units and relative clock skews. In order for this to

be accurate, the packets carrying the different TS values should experience equal (or at least similar delay) when traveling from the sender to the receiver, as any difference in the experienced delays would introduce error in the unit/skew estimation. The receiver should then choose two packets that have experienced a similar delay (for example, the minimum delay of n packets). The problem is that the delay measured is contaminated by the error in the clock unit/skew estimation, so it is not possible to tell which is the packet that experienced the minimum delay. It would be possible to select the packets that experienced a minimum RTT. The problem with this approach is that the error is essentially the RTT measured, which is not an acceptable bound.

Moreover, as this measure to estimate the Time Stamp clock units and drift is affected by the propagation time of the packets themselves, it is not possible to estimate this value by just using two packets if the time between them is short [khono]). Although better estimations can be achieved from using multiple points to estimate the value (i.e., through lineal regression), a non negligible time is required until enough data is gathered to reach the required precission.

An additional difficulty regarding the estimation of the TS units and clock skew in the context of (r)LEDBAT is that the LEDBAT congestion controller actions directly affect the (queueing) delay experienced by packets. In particular, if there is an error in the estimation of the TS units/skew, the LEDBAT controller will attempt to compensate it by reducing/increasing the load. The result is that the LEDBAT operation interferes with the TS units/clock skew measurements. Because of this, measurements are more accurate when there is no traffic in the connection (in addition to the packets used for the measurements). The problem is that the receiver is unaware if the sender is injecting traffic at any point in time, and opportunistically seize quiet intervals to preform measurements. The receiver can however, force periodic slowdowns, reducing the announced receive window to a few packets and perform the measurements then.

Authors' Addresses

Marcelo Bagnulo
UC3M

Email: marcelo@it.uc3m.es

Alberto Garcia-Martinez
UC3M

Email: alberto@it.uc3m.es

Gabriel Montenegro
Microsoft

Email: Gabriel.Montenegro@microsoft.com

Praveen Balasubramanian
Microsoft

Email: pravb@microsoft.com