Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 6 August 2025

M. Bagnulo
A. Garcia-Martinez
Universidad Carlos III de Madrid
G. Montenegro

P. Balasubramanian
Confluent

2 February 2025

rLEDBAT: receiver-driven Low Extra Delay Background Transport for TCP
draft-irtf-iccrg-rledbat-10

Abstract

This document specifies rLEDBAT, a set of mechanisms that enable the execution of a less-than-best-effort congestion control algorithm for TCP at the receiver end. This document is a product of the Internet Congestion Control Research Group (ICCRG) of the Internet Research Task Force (IRTF).

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 6 August 2025.

Table of Contents

1. Introduction 2
2. Motivations for rLEDBAT 3
3. rLEDBAT mechanisms 4
3.1. Controlling the receive window 6
3.1.1. Avoiding window shrinking 7
3.1.2. Setting the Window Scale Option 8
3.2. Measuring delays 8
3.2.1. Measuring RTT to estimate the queueing delay 9
3.2.2. Measuring one way delay to estimate the queueing delay 11
3.3. Detecting packet losses and retransmissions 13
4. Experiment Considerations 13
4.1. Status of the experiment at the time of this writing. . . 14
5. Security Considerations 15
6. IANA Considerations 16
7. Acknowledgements 16
8. Informative References 16
Appendix A. Terminology 17
Appendix B. rLEDBAT pseudo-code 18
Authors' Addresses 20

1. Introduction

LEDBAT (Low Extra Delay Background Transport) [RFC6817] is a congestion-control algorithm used for less-than-best-effort (LBE) traffic.

When LEDBAT traffic shares a bottleneck with other traffic using standard congestion control algorithms (for example, TCP traffic using Cubic[RFC9438], hereafter referred as standard-TCP for short), it reduces its sending rate earlier and more aggressively than standard-TCP congestion control, allowing other non-background traffic to use more of the available capacity. In the absence of competing 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. With respect 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.irtf-iccrg-ledbat-plus-plus] is also an LBE congestion control algorithm which is inspired by 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 specification, we describe rLEDBAT, a set of mechanisms that enable the execution of an LBE delay-based congestion control algorithm such as LEDBAT or LEDBAT++ at the receiver end of a TCP connection.

The consensus of the Internet Congestion Control Research Group (ICCRG) is to publish this document to encourage further experimentation and review of rLEDBAT. This document is not an IETF product and is not a standard. The status of this document is experimental. In section 4 titled Experiment Considerations, we describe the purpose of the experiment and its current status.

2. Motivations for rLEDBAT

rLEDBAT enables new use cases and new deployment models, fostering the use of LBE traffic. 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 that does not implement 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 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 by the receiver 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 [RFC9293] 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 [RFC9438] or New Reno [RFC5681].

rLEDBAT does not define 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 [RFC6817] or the LEDBAT++ congestion control algorithm [I-D.irtf-iccrg-ledbat-plus-plus]. The rLEDBAT MAY use other LBE congestion control algorithms defined elsewhere. Irrespective 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 packet delay (either the round trip time or the one way delay, depending on the selected algorithm), mechanisms to detect packet loss and the means to manipulate the Receive Window to control the sender’s rate. The former provide input to the LBE congestion control algorithm while the latter uses the congestion window computed by the LBE congestion control algorithm to manipulate the Receive window, as depicted in the figure.

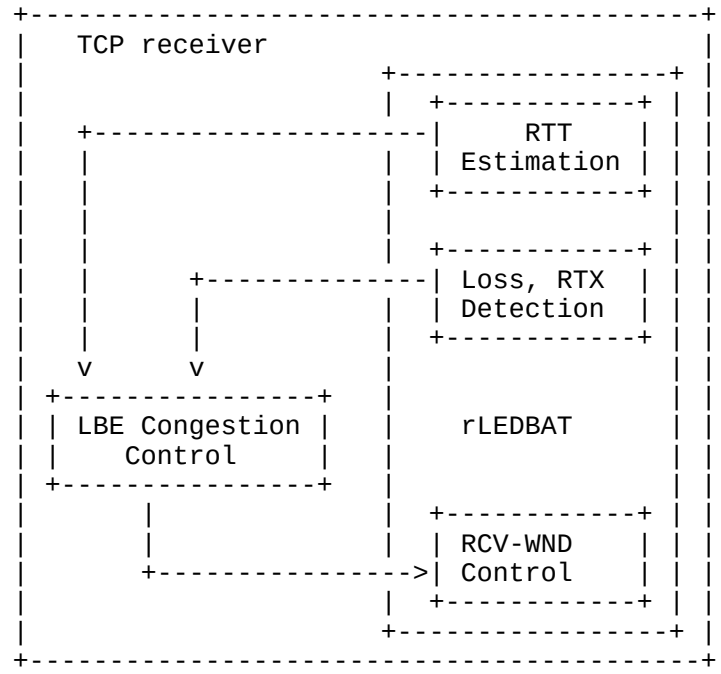

Figure 1: The rLEDBAT architecture.

We describe each of the rLEDBAT components 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. [RFC9293] 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 fcwnd the value that a standard RFC793bis TCP receiver calculates to set in the receive window for flow control purposes. We call RLWND 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 == fcwnd.

In the case of rLEDBAT receiver, the rLEDBAT receiver MUST NOT set the RCV.WND to a value larger than fcwnd and it SHOULD set the RCV.WND to the minimum of RLWND and fcwnd, 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 standard-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 RLWND 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. One exception to this is at the beginning of the connection, when there is no information to set RLWND, then, RLWND is set to its maximum value, so that the sending rate of the sender is governed by the flow control algorithm of the receiver and the TCP slow start mechanism of the sender.

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

3.1.1. Avoiding window shrinking

The LEDBAT/LEDBAT++ algorithm executed in a rLEDBAT receiver increases or decreases RLWND according to congestion signals (variations on the estimated queueing delay and packet loss). If RLWND is decreased and directly announced in RCV.WND, this could lead to an announced window that is smaller than what is currently in use. This so called 'shrinking the window' is discouraged as per [RFC9293], as it may cause unnecessary packet loss and performance penalty. To be consistent with [RFC9293], the rLEDBAT receiver SHOULD NOT shrink the receive window.

In order to avoid window shrinking, the receiver MUST only reduce RCV.WND by the number of bytes upon of a received data packet. This may fall short to honor the new calculated value of the RLWND immediately. However, 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 sufficient to honor the window output from the LEDBAT/LEDBAT++ algorithms since they only allow to perform at most one multiplicative decrease per RTT.

3.1.2. Setting the Window Scale Option

The Window Scale (WS) option [RFC7323] is a means to increase the maximum window size permitted by the Receive Window. The WS option defines a scale factor which restricts the granularity of the receive window that can be announced. This means that the rLEDBAT client will have to accumulate the increases resulting from multiple received packets, and only convey a change in the window when the accumulated sum of increases is equal or higher than one increase step as imposed by the scaling factor 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's segmentation practice results in sending full segments (i.e., segments of size equal to the MSS). 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 (unless packets smaller than the MSS are being exchanged).

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 on 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 is set independently for each direction of the TCP connection.

3.2. Measuring delays

Both LEDBAT and LEDBAT++ measure base and current delays to estimate the queueing delay. LEDBAT uses the one way delay while LEDBAT++ uses the round trip time. In the next sections we describe how rLEDBAT mechanisms enable the receiver to measure the one way delay or the round trip time, whatever is needed depending on the congestion control algorithm used.

3.2.1. Measuring 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 RTT, the rLEDBAT receiver estimates the base RTT (i.e., the constant components of 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 RTT to estimate the queueing delay has a number of shortcomings and difficulties that we discuss next.

The queuing delay measured using 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 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 RTT. This allows the rLEDBAT receiver to measure 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 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 RTT and sampling a subset of in-flight packets per RTT is enough to properly assess the queueing delay. 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 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 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 the sender seldomly will have to wait until the next block is available to proceed with the data transfer. To address this situation, the filter used by the congestion control algorithm executed in the receiver SHOULD discard outliers (e.g. a min filter would achieve this) when measuring RTT using pure ACK packets.

This limitation of the sender’s window 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 node 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 RTT measurements during the window reduction phase that are triggered by pure ACK packets. 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 RTT when sending data packets

In the case that the rLEDBAT node is sending data packets and matching them with pure ACKs to measure 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 RTT measurements 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, measurements based on data packets from the rLEDBAT node matching pure ACKs from the other end will result in an increased RTT sample. The additional increase in the measured RTT will be up to 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 [RFC9293]. The rLEDBAT receiver MAY discard 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 discards outliers would also address this issue (e.g. a min filter).

3.2.2. Measuring one way delay to estimate the queueing delay

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 TimeStamp option to measure the one way delay by subtracting the timestamp contained in the incoming packet from the local time at which the packet has arrived. As noted in [RFC6817] the clock offset between the clock of

the sender and the clock in the receiver does not affect the LEDBAT operation, since LEDBAT uses the difference between the base one way delay and the current one way delay to estimate the queuing delay, effectively canceling the clock offset error in the queueing delay estimation. There are however two other issues that the rLEDBAT receiver needs to take into account in order to properly estimate the one way delay, namely, the units in which the received timestamps are expressed and the clock skew. We address them next.

In order to measure the one way delay using TCP timestamps, the rLEDBAT receiver, first, needs to discover the units of values in the TS option and, second, needs to account for the skew between the two endpoint clocks. Note that a mismatch of 100 ppm (parts per million) in the estimation of the sender’s clock rate accounts for 6 ms of variation per minute in the measured delay. This just one order of magnitude below the target delay set by rLEDBAT (or potentially more if the target is set to lower values, which is possible). 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 measures how much local time has elapsed between two packets with different TS values issued by the sender. 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. One possible approach is to select packets that experienced the minimum delay (i.e. close to zero queueing delay) to make the estimations.

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 so, it is unable to use these quiet intervals to perform measurements. The receiver can however, force periodic slowdowns, reducing the announced receive window to a few packets and perform the measurements then.

It is possible for the rLEDBAT receiver to perform multiple measurements to assess both the TS units and the relative clock skew during the lifetime of the connection, in order to obtain more accurate results. Clock skew measurements are more accurate if the time period used to discover the skew is larger, as the impact of the skew becomes more apparent. It is a reasonable approach for the rLEDBAT receiver to perform an early discovery of the TS units (and the clock skew) using the first few packets of the TCP connection and then improve the accuracy of the TS units/clock skew estimation using periodic measurements later in the lifetime of the connection.

3.3. Detecting packet losses and retransmissions

The rLEDBAT receiver is capable of detecting retransmitted packets in the following way. We call RCV.HGH the highest sequence number corresponding 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 reducing 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. Experiment Considerations

The status of this document is Experimental. The general purpose of the proposed experiment is to gain more experience running rLEDBAT over different network paths to see if the proposed rLEDBAT parameters perform well in different situations. Specifically, we would like to learn about the following aspects of the rLEDBAT mechanism:

- Interaction between the sender and the receiver Congestion control algorithms. rLEDBAT posits that because the rLEDBAT receiver is using a less-than-best-effort congestion control algorithm, the receiver congestion control algorithm will expose a smaller congestion window (conveyed though the Receive Window) than the one resulting from the congestion control algorithm executed at the sender. One of the purposes of the experiment is learn how these two interact and if the assumption that the receiver side is always controlling the sender's rate (and making rLEDBAT effective) holds. The experiment should include the different congestion control algorithms that are currently widely used in the Internet, including Cubic, BBR and LEDBAT(++).

- Interaction between rLEDBAT and Active Queue Management techniques such as Codel, PIE and L4S.

- How the rLEDBAT should resume after a period during which there was no incoming traffic and the information about the rLEDBAT state information is potentially dated.

4.1. Status of the experiment at the time of this writing.

Currently there are the following implementations of rLEDBAT that can be used for experimentation:

- Windows 11. rLEDBAT is available in Microsoft's Windows 11 22H2 since October 2023 [Windows11].

- Windows Server 2022. rLEDBAT is available in Microsoft's Windows Server 2022 since September 2022 [WindowsServer].

- Apple. rLEDBAT is available in MacOS and iOS since 2021 [Apple].

- Linux implementation, open source, available since 2022 at https://github.com/net-research/rledbat_module.

- ns3 implementation, open source, available since 2020 at https://github.com/manas11/implementation-of-rLEDBAT-in-ns-3.

In addition, rLEDBAT has been deployed by Microsoft in wide scale in the following services:

- BITS (Background Intelligent Transfer Service)

- DO (Delivery Optimization) service

- Windows update # using DO

- Windows Store # using DO

- OneDrive

- Windows Error Reporting # wermgr.exe; werfault.exe

- System Center Configuration Manager (SCCM)

- Windows Media Player

- Microsoft Office

- Xbox (download games) # using DO

Some initial experiments involving rLEDBAT have been reported in [COMNET3]. Experiments involving the interaction of LEDBAT++ and BBR are presented in [COMNET2]. An experimental evaluation of the LEDBAT++ algorithm is presented in [COMNET1]. As LEDBAT++ is one of the less-than-best-effort congestion control algorithms that rLEDBAT relies on, the results regarding LEDBAT++ interaction with other congestion control algorithms are relevant for the understanding of rLEDBAT as well.

5. Security Considerations

Overall, we believe that rLEDBAT does not introduce any new vulnerabilities to existing TCP endpoints, as it relies on existing TCP knobs, notably the Receive Window and timestamps.

Specifically, rLEDBAT uses RCV.WND to modulate the rate of the sender. An attacker wishing to starve a flow can simply reduce the RCV.WND, irrespective of whether rLEDBAT is being used or not.

We can further ask ourselves whether the attacker can use the rLEDBAT mechanisms in place to force the rLEDBAT receiver to reduce the RCV WND. There are two ways an attacker can do that. One would be to introduce an artificial delay to the packets either by actually delaying the packets or modifying the Timestamps. This would cause the rLEDBAT receiver to believe that a queue is building up and reduce the RCV.WND. Note that an attacker to do that must be on path, so if that is the case, it is probably more direct to simply reduce the RCV.WND.

The other option would be for the attacker to make the rLEDBAT receiver believe that a loss has occurred. To do that, it basically needs to retransmit an old packet (to be precise, it needs to transmit a packet with the right sequence number and the right port and IP numbers). This means that the attacker can achieve a

reduction of incoming traffic to the rLEDBAT receiver not only by modifying the RCV.WND field of the packets originated from the rLEDBAT host, but also by injecting packets with the proper sequence number in the other direction. This may slightly expand the attack surface.

6. IANA Considerations

No actions are required from IANA.

7. Acknowledgements

This work was supported by the EU through the StandICT projects RXQ, CCI and CEL6, the NGI Pointer RIM project and the H2020 5G-RANGE project and by the Spanish Ministry of Economy and Competitiveness through the 5G-City project (TEC2016-76795-C6-3-R).

We would like to thank ICCRG chairs Reese Enghardt and Vidhi Goel for their support on this work. We would also like to thank Daniel Havey for his help. We would like to thank Colin Perkins, Mirja Kuehlewind, and Vidhi Goel for their reviews and comments on earlier versions of this document.

8. Informative References

[Apple] Stuart, S.C. and V.G. Vidhi, "Reduce network delays for your app", WWDC21 https://developer.apple.com/videos/play/wwdc2021/10239/, 2021.

[COMNET1] Bagnulo, M.B. and A.G. Garcia-Martinez, "An experimental evaluation of LEDBAT++", Computer Networks Volume 212, 2022.

[COMNET2] Bagnulo, M.B. and A.G. Garcia-Martinez, "When less is more: BBR versus LEDBAT++", Computer Networks Volume 219, 2022.

[COMNET3] Bagnulo, M.B., Garcia-Martinez, A.G., Mandalari, A.M., Balasubramanian, P.B,., Havey, D.H., and G.M. Montenegro, "Design, implementation and validation of a receiver-driven less-than-best-effort transport", Computer Networks Volume 233, 2022.

[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://datatracker.ietf.org/doc/html/draft-irtf-iccrg-ledbat-plus-plus-01>.

[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>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC9438] Xu, L., Ha, S., Rhee, I., Goel, V., and L. Eggert, Ed., "CUBIC for Fast and Long-Distance Networks", RFC 9438, DOI 10.17487/RFC9438, August 2023, <https://www.rfc-editor.org/info/rfc9438>.

[Windows11]
Forsmann, C.F., "What’s new in Delivery Optimization", Microsoft Documentation https://learn.microsoft.com/en-us/windows/deployment/do/whats-new-do, 2023.

[WindowsServer]
Havey, D.H., "LEDBAT Background Data Transfer for Windows", Microsoft Blog https://techcommunity.microsoft.com/t5/networking-blog/ledbat-background-data-transfer-for-windows/ba-p/3639278, 2022.

Appendix A. Terminology

We use the following abreviations thoughout the text. We include a short list for the reader’s convenence:

RCV.WND: the value included in the Receive Window field of the TCP header (which computation is modified by this specification)

SND.WND: The TCP sender’s window

cwnd: the consgestion window as computed by the congestion control algorithm running at the TCP sender.

RLWND: the window value calculated by rLEDBAT algorithm

fcwnd: the value that a standard RFC793bis TCP receiver calculates to set in the receive window for flow control purposes.

RCV.HGH: the highest sequence number corresponding to a received byte of data at one point in time

TSV.HGH: TSV.HGH the TSVal value corresponding to the segment in which RCV.HGH was carried at that point in time

SEG.SEQ: the sequence number of the last received segment

TSV.SEQ: the TSVal value of the last received segment

Appendix B. rLEDBAT pseudo-code

We next describe how to integrate the proposed rLEDBAT mechanisms and an LBE delay-based congestion control algorithm such as LEDBAT or LEDBAT++. We describe the integrated algorithm as two procedures, one that is executed when a packet is received by a rLEDBAT-enabled endpoint (Figure 2) and another that is executed when the rLEDBAT-enabled endpoint sends a packet (Figure 3). At the beginning, RLWND is set to its maximum value, so that the sending rate of the sender is governed by the flow control algorithm of the receiver and the TCP slow start mechanism of the sender, and the ackedBytes variable is set to 0.

We assume that the LBE congestion control algorithm defines a WindowIncrease() function and a WindowDecrease() function. For example, in the case of LEDBAT++, the WindowIncrease() function is an additive increase, while the WindowDecrease() function is a multiplicative decrease. In the case of the WindowIncrease(), we assume that it takes as input the current window size and the number of bytes that were acknowledged since the last window update (ackedBytes) and returns as output the updated window size. In the case of WindowDecrease(), it takes as input the current window size and returns the updated window size.

The data structures used in the algorithms are as follows. The sentList is a list that contains the TSval and the local send time of each packet sent by the rLEDBAT-enabled endpoint. The TSecr field of the packets received by the rLEDBAT-enabled endpoint are matched with the sendList to compute the RTT.

The RTT values computed for each received packet are stored in the RTTlist, which contains also the received TSecr (to avoid using multiple packets with the same TSecr for RTT calculations, only the first packet received for a given TSecr is used to compute the RTT). It also contains the local time at which the packet was received, to allow selecting the RTTs measured in a given period (e.g., in the last 10 minutes). RTTlist is initialized with all its values to its maximum.

```
procedure receivePacket()
  //Looks for first sent packet with same TSval as TSecr, and,
  //returns time difference
  receivedRTT = computeRTT(sentList, receivedTSecr, receivedTime)

  //Inserts minimum value for a given receivedTSecr
  //note that many received packets may contain same receivedTSecr
  insertRTT (RTTlist, receivedRTT, receivedTSecr, receivedTime)

  filteredRTT = minLastKMeasures(RTTlist, K=4)
  baseRTT = minLastNSeconds(RTTlist, N=180)
  qd = filteredRTT - baseRTT

  //ackedBytes is the number of bytes that can be used to reduce
  //the Receive Window - without shrinking it - if necessary
  ackedBytes = ackedBytes + receiveBytes

  if retransmittedPacketDetected then
        RLWND = DecreaseWindow(RLWND) // Only once per RTT
  end if
  if qd < T then
        RLWND = IncreaseWindow(RLWND, ackedBytes)
  else
        RLWND = DecreaseWindow(RLWND)
  end if
end procedure
```

Figure 2: Procedure executed when a packet is received

```
   procedure SENDPACKET
     if (RLWND > RLWNDPrevious) or (RLWND - RLWNDPrevious < ackedBytes)
     then
           RLWNDPrevious = RLWND
     else
           RLWNDPrevious = RLWND - ackedBytes
     end if
     ackedBytes = 0
     RLWNDPrevious = RLWND

     //Compute the RWND to include in the packet
     RLWND = min(RLWND, fcwnd)
   end procedure
```

Figure 3: Procedure executed when a packet is sent

Authors' Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Email: marcelo@it.uc3m.es

Alberto Garcia-Martinez
Universidad Carlos III de Madrid
Email: alberto@it.uc3m.es

Gabriel Montenegro
Email: g.e.montenegro@hotmail.com

Praveen Balasubramanian
Confluent
Email: pravb.ietf@gmail.com