

Network Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: May 1, 2020

M. Bagnulo  
A. Garcia-Martinez  
UC3M  
G. Montenegro  
P. Balasubramanian  
Microsoft  
October 29, 2019

rLEDBAT: receiver-driven Low Extra Delay Background Transport for TCP  
draft-bagnulo-iccrgr-ledbat-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

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 May 1, 2020.

## Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Motivations for rLEDBAT . . . . .	3
3. rLEDBAT mechanisms . . . . .	4
3.1. Controlling the receive window . . . . .	5
3.1.1. Avoiding window shrinking . . . . .	6
3.1.2. Window Scale Option . . . . .	7
3.2. Measuring the Round Trip Time . . . . .	7
3.2.1. Measuring the RTT to estimate the queueing delay . . . . .	8
3.3. Detecting retransmissions and packet losses . . . . .	10
4. Security Considerations . . . . .	11
5. IANA Considerations . . . . .	11
6. Acknowledgements . . . . .	11
7. Informative References . . . . .	11
Appendix A. Difficulties while measuring one way delay at the TCP receiver . . . . .	12
Authors' Addresses . . . . .	13

## 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 queueing 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 queueing 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-iccr-leadbatplusplus] 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-iccr-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?

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 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:  $\text{SND.WND} = \min(\text{cwnd}, \text{rl.WND}, \text{fc.WND})$

#### 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 queueing 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 queueing delay to be used by the rLEDBAT controller.

LEDBAT++ discovers the base RTT (RTT<sub>b</sub>) by taking the minimum value of the measured RTTs over a period of time. The current RTT (RTT<sub>c</sub>) 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 queueing 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 TS<sub>Val</sub> value carried in outgoing packets with the TS<sub>Secr</sub> 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 TS<sub>Val</sub> value. If that happens, the rLEDBAT receiver will be unable to match the different outgoing packets carrying the same TS<sub>Val</sub> value with the different incoming packets carrying also the same TS<sub>Secr</sub> 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 TSVval 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-iccr-geledbatplusplus]  
Balasubramanian, P., Ertugay, O., and D. Havey, "LEDBAT++:  
Congestion Control for Background Traffic", draft-  
balasubramanian-iccr-geledbatplusplus-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 precision.

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

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 7, 2020

P. Balasubramanian  
O. Ertugay  
D. Havey  
Microsoft  
November 4, 2019

LEDBAT++: Congestion Control for Background Traffic  
draft-balasubramanian-iccr-ledbatplusplus-01

Abstract

This experimental memo describes LEDBAT++, a set of enhancements to the LEDBAT (Low Extra Delay Background Transport) congestion control algorithm for background traffic. The LEDBAT congestion control algorithm has several shortcomings that prevent it from working effectively in practice. LEDBAT++ extends LEDBAT by adding a set of improvements, including reduced congestion window gain, modified slow-start, multiplicative decrease and periodic slowdowns. This set of improvement mitigates the known issues with the LEDBAT algorithm, such as latency drift, latecomer advantage and inter-LEDBAT fairness. LEDBAT++ has been implemented as a TCP congestion control algorithm in the Windows operating system. LEDBAT++ has been deployed in production at scale on a variety of networks and been experimentally verified to achieve the original stated goals of LEDBAT.

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 May 7, 2020.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. LEDBAT Issues . . . . .	3
3.1. Latecomer advantage . . . . .	3
3.2. Inter-LEDBAT fairness . . . . .	4
3.3. Latency drift . . . . .	4
3.4. Low latency competition . . . . .	4
3.5. Dependency on one-way delay measurements . . . . .	5
4. LEDBAT++ Mechanisms . . . . .	5
4.1. Modified slow start . . . . .	5
4.2. Slower than Reno increase . . . . .	5
4.3. Multiplicative decrease . . . . .	6
4.4. Initial and periodic slowdown . . . . .	7
4.5. Use of Round Trip Time instead of one way delay . . . . .	7
5. Deployment Issues . . . . .	8
6. Security Considerations . . . . .	8
7. IANA Considerations . . . . .	8
8. Acknowledgements . . . . .	8
9. References . . . . .	9
9.1. Normative References . . . . .	9
9.2. Informative References . . . . .	9
Authors' Addresses . . . . .	9

## 1. Introduction

Operating systems and applications use background connections for a variety of tasks, such as software updates, large media downloads, telemetry, or error reporting. These connections should operate without affecting the general usability of the system. Usability is measured in terms of available network bandwidth and network latency. LEDBAT [RFC6817] is designed to minimize the impact of lower than best effort connections on the latency and bandwidth of other connections. To achieve that, each LEDBAT connection monitors the transmission delay of packets, and compares them to the minimum delay observed on the connection. The difference between the transmission delay and the minimum delay is used as an estimate of the queuing



delay. If the queuing delay is above a target, LEDBAT directs the connection to reduce its bandwidth. If the queuing delay is below the target, the connection is allowed to increase its transmission rate. The bandwidth increase and decrease are proportional to the difference between the observed values and the target. LEDBAT reacts to packet losses and other congestion signals in the same way as standard TCP.

However, there are a few issues that plague LEDBAT, some previously documented, and some discovered by experiments. LEDBAT++ adds additional mechanisms on top of (and in some cases deviates from) LEDBAT to overcome these problems. The remaining sections describe the problems and the mechanisms in detail. The objective of this informational RFC is to document LEDBAT++ enhancements on top of a base LEDBAT implementation in the Windows operating system. encourage its use so the algorithm can be further verified and improved.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. LEDBAT Issues

This section lists some known LEDBAT issues from existing literature and also list some new problems observed as a result of experimentation with an implementation of [RFC6817].

### 3.1. Latecomer advantage

Delay based congestion control protocols like LEDBAT are known to suffer from a latecomer advantage. When the newcomer establishes a connection, the transmission delay that it encounters incorporates queuing delay caused by the existing connections. The newcomer considers this large delay the minimum, and thereby increases its transmission rate while other LEDBAT connections slow down. Eventually, the latecomer will end up using the entire bandwidth of the connection. Standard TCP congestion control as described in [RFC0793] and [RFC5681], causes some queuing, the LEDBAT delay measurements incorporate that queuing, and the base delay as measured by the connection is thus set to a larger value than the actual minimum. As a result, the queues remain mostly full. In some cases, this queuing persists even after the closing of the competing TCP connection. This phenomenon was already known during the design of LEDBAT, but there is no mitigation in the LEDBAT design. The designers of the protocol relied instead on the inherent burstiness of network traffic. Small gaps in transmission schedules would allow

the latecomer to measure the true delay of the connection. This reasoning is not satisfactory because workloads can upload large amount of data, and would not always see such gaps.

### 3.2. Inter-LEDBAT fairness

The latecomer advantage is caused by the improper evaluation of the base delay, with the latecomer using a larger value than the preexisting connections. However, even when all competing connections have a correct evaluation of the base delay, some of them will receive a larger share of resource. The reason for that persistent unfairness is explained in [RethinkLEDBAT]. LEDBAT specifies proportional feedback based on a ratio between the measured queuing delay and a target. Proportional feedback uses both additive increases and additive decreases. This does stabilize the queue sizes, but it does not guarantee fair sharing between the competing connections.

### 3.3. Latency drift

LEDBAT estimates the base delay of a connection as the minimum of all observed transmission delays over a 10-minute interval. It uses an interval rather than a measurement over the whole duration of the connection, because network conditions may change over time. For example, an existing connection may be transparently rerouted over a longer path, with a longer transmission delay. Keeping the old estimate would then cause LEDBAT to unnecessarily reduce the connection throughput. However experiments show that this causes a ratcheting effect when LEDBAT connections are allowed to operate for a long time. The delay feedback in LEDBAT causes the queuing delay to stabilize just below the target. After an initial interval, all new measurements are equal to the initial transmission delay plus a fraction of the target. Every 10 minutes, the measured base delay increases by that fraction of the target queuing delay, leading to potentially large values over time.

### 3.4. Low latency competition

LEDBAT compares the observed queuing delays to a fixed target. The target value cannot be set too low, because that would cause poor operation on slow networks. In practice, it is set to 60ms, a value that allows proper operation of latency sensitive applications like VoIP. But if the bottleneck buffer is small such that the queuing delay will never reach the target, then the LEDBAT connection behaves just like an ordinary connection. It competes aggressively, and obtains the same share of the bandwidth as regular TCP connections. On high speed links the problem is exacerbated.

### 3.5. Dependency on one-way delay measurements

The LEDBAT algorithm requires use of one-way delay measurements. This makes it harder to use with transport protocols like TCP that have no reliable way to obtain one way delay measurements. TCP timestamps do not standardize clock frequency, and the endpoints will need to rely on heuristics to guess the clock frequency of the remote peer to detect and correct for clock skew. TCP timestamps do not include clock synchronization, and would need some non-standard invention to compensate for clock skew. Any such mechanism is very fragile.

## 4. LEDBAT++ Mechanisms

### 4.1. Modified slow start

Traditional initial slow start can cause spikes in bandwidth usage. However skipping exponential congestion window increase results in really poor performance on long delay links. LEDBAT++ applies the dynamic GAIN parameter to the congestion window increases. In standard TCP operation, the congestion window increases for every ACK by exactly the amount of bytes acknowledged. A LEDBAT++ sender increases the congestion window by that number multiplied by the dynamic GAIN value. In low latency links, this ensures that LEDBAT++ connections ramp up slower than regular connections. LEDBAT++ sender limits the initial window to 2 packets. LEDBAT++ sender monitors the transmission delays during the slow start period. If the queuing delay is larger than 3/4ths of the target delay, exit slow start and immediately move to the congestion avoidance phase. After initial slow start, the increase of congestion window is bounded by the Ssthresh estimate acquired during congestion avoidance, and the risk of creating congestion spikes is very low. Exit slow start in on excessive delay SHOULD be applied only during the initial slow start.

### 4.2. Slower than Reno increase

When the queuing delays are below the target delay, LEDBAT behaves like standard TCP [RFC0793]. LEDBAT introduces a GAIN parameter which can be set between 0 and 1. In order to solve the low latency competition problem, LEDBAT++ makes the GAIN parameter dynamic. When standard and reduced connections share the same bottleneck, they experience the same packet drop rate. The GAIN value ensures that the throughput of the LEDBAT connection will be a fraction ( $1/\sqrt{1/\text{GAIN}}$ ) of the throughput of the regular connections. Small values of GAIN work well when the base delay is small, and ensure that the LEDBAT connection will yield to regular connections in these networks. However, large values of GAIN do not work well on long delay links. In the absence of competing traffic, combining large

base delays with small GAIN values causes the connection bandwidth to remain well under capacity for a long time. In LEDBAT++, GAIN is a function of the ratio between the base delay and the target delay:

$$\text{GAIN} = 1 / (\min(16, \text{CEIL}(2 * \text{TARGET} / \text{base})))$$

where CEIL(X) is defined as the smallest integer larger than X. Implementations MAY experiment with the constant value 16 as a tradeoff between responsiveness and performance.

#### 4.3. Multiplicative decrease

[RethinkLEDBAT] suggests combining additive increases and multiplicative decreases in order to solve the Inter-LEDBAT fairness problem. It proposes to change the way LEDBAT increases and decreases the congestion window based on the ratio between the observed delay and the target. Assuming that the congestion window is changed once per roundtrip measurement. In standard LEDBAT, the per RTT window when delay is less than target is:

$$W += \text{GAIN} * (1 - \text{delay} / \text{target})$$

In LEDBAT++, with multiplicative decrease, the per RTT window when delay is less than target is:

$$W += \text{GAIN}$$

Similarly in standard LEDBAT, the per RTT window when the delay is higher than target is:

$$W -= \text{GAIN} * (\text{delay} / \text{target} - 1)$$

In LEDBAT++, with multiplicative decrease, the per RTT window delay is higher than target is:

$$W += \max((\text{GAIN} - \text{Constant} * W * (\text{delay} / \text{target} - 1)), -W/2)$$

It is RECOMMENDED that the Constant be set to 1. Implementations MAY experiment with this value. If the connections have different estimates of the base delay, capping the multiplicative decrease to at most W/2 is required. Otherwise, spikes in delay can cause the window to immediately drop to its minimal value. LEDBAT++ sender MUST also ensure that the congestion window never decreases below 2 packets, in order to avoid completely starving the connection.

#### 4.4. Initial and periodic slowdown

The LEDBAT specification assumes that there will be natural gaps in traffic, and that during those gaps the observed delay corresponds to a state where the queues are empty. However, there are workloads where the traffic is sustained for long periods. This causes base delay estimates to be inaccurate and is one of the major reasons behind latency drift as well as the lack of inter-LEDBAT fairness. To ensure stability, LEDBAT++ forces these gaps, or slow down periods. A slowdown is an interval during which the LEDBAT++ connection voluntarily reduces its traffic, allowing queues to drain and transmission delay measurements to converge to the base delay. The slowdown works as follows:

- o Upon entering slowdown, set SSTHRESH to the current version of the congestion window CWND, and then reduce CWND to 2 packets.
- o Keep CWND frozen at 2 packets for 2 RTT.
- o After 2 RTT, ramp up the congestion window according to the slow start algorithm, until the congestion window reaches SSTHRESH.

Keeping the CWND frozen at 2 packets for 2 RTT allows the queues to drain, and is key to obtaining accurate delay measurements. The initial slowdown starts shortly after the connection completes the initial slow start phase; 2 RTT after the initial slow start completes. After the initial slowdown, LEDBAT++ sender performs periodic slowdowns. The interval between slowdown is computed so that slowdown does not cause more than a 10% drop in the utilization of the bottleneck. LEDBAT++ sender measures the duration of the slowdown, from the time of entry to the time at which the congestion window regrows to the previous SSTHRESH value. The next slowdown is then scheduled to occur at 9 times this duration after the exit point. The combination of initial and periodic slowdowns allows competing LEDBAT connections to obtain good estimates of the base delay, and when combined with multiplicative decrease solves both the latecomer advantage and the Inter-LEDBAT fairness problems.

#### 4.5. Use of Round Trip Time instead of one way delay

LEDBAT++ uses Round Trip Time measurements instead of one way delay. One possible shortcoming of round trip delay measurements is that they incorporate queuing delays in both directions. This can lead to unnecessary slowdowns, such as slowing down an upload connection because a download is saturating the downlink but in practice this seems to benefit the workloads because bottleneck link can carry ACK traffic in the other direction for the competing flows. Round trip measurements also include the delay at the receiver between receiving

a packet and sending the corresponding acknowledgement. These delays are normally quite small, except when the delayed acknowledgment logic kicks in. Effect of delayed ACK can be particularly acute when the congestion window only includes a few packets, for example at the beginning of the connection.

The problems of using one way delay are mitigated through a set of implementation choices. First, LEDBAT++ sender enables the TCP Timestamp option, in order to obtain RTT samples with each acknowledgement. A LEDBAT++ sender SHOULD filter the round trip measurements by using the minimum of the 4 most recent delay samples, as suggested in the LEDBAT specification. Finally, the queueing delay target is set larger than the typical TCP maximum acknowledgement delay. This avoids over reacting to a single delayed ACK measurement. LEDBAT++ default delay target of 60ms is different from the 100ms value recommended in [RFC6817].

## 5. Deployment Issues

LEDBAT++ is a sender-side algorithmic improvement. This implies that for many workloads it requires changes to the servers serving content. It does not address workloads or scenarios where the only entities that can be updated are clients.

Transparent proxies prevent measurement of end-to-end delay and might interfere with the effective operation of LEDBAT++.

The interaction between Active Queue Management (AQM) and LEDBAT++ is an area of research.

## 6. Security Considerations

LEDBAT++ enhances LEDBAT and inherits the general security considerations discussed in [RFC6817].

## 7. IANA Considerations

This document has no actions for IANA.

## 8. Acknowledgements

The LEDBAT++ algorithm was designed and implemented by Osman Ertugay, Christian Huitema, Praveen Balasubramanian, and Daniel Havey.

## 9. References

### 9.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [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>>.

### 9.2. Informative References

- [RethinkLEDBAT] Carofiglios, G., Muscariello, L., Rossi, D., Testa, C., and S. Valenti, "Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol", Computer Networks, Volume 57, Issue 8, 4 June 2013, Pages 1838-1852, 2013, <<http://perso.telecom-paristech.fr/~drossi/paper/rossi13comnet.pdf>>.

### Authors' Addresses

Praveen Balasubramanian  
Microsoft  
One Microsoft Way  
Redmond, WA 98052  
USA

Phone: +1 425 538 2782  
Email: [pravb@microsoft.com](mailto:pravb@microsoft.com)

Osman Ertugay  
Microsoft

Phone: +1 425 706 2684  
Email: osmaner@microsoft.com

Daniel Havey  
Microsoft

Phone: +1 425 538 5871  
Email: dahavey@microsoft.com



TSVWG  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2020

R. Even  
Huawei  
R. Huang  
Huawei Technologies Co., Ltd.  
October 23, 2019

Data Center Fast Congestion Management  
draft-even-iccr-g-dc-fast-congestion-00

Abstract

Fast congestion control is discussed in academic papers as well as in the different standard bodies. There is no one proposal for providing a solution that will work for all use cases leading to multiple approaches. By congestion control we refer to an end to end solution and not only to the congestion control algorithm on the sender side. This document describes the current state of flow control and congestion for Data Centers and proposes future directions.

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 April 25, 2020.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Conventions . . . . .	3
3. Abbreviations . . . . .	3
4. Alternative Congestion Management mechanisms . . . . .	4
4.1. Mechanisms based on estimation of network status . . . . .	4
4.2. Network provides limited information . . . . .	4
4.2.1. ECN and DCTCP . . . . .	5
4.2.2. DCQCN . . . . .	5
4.2.3. SCE – Some Congestion Experienced . . . . .	6
4.2.4. L4S – Low Latency, Low Loss, Scalable Throughput . . . . .	7
4.3. Network provides more information . . . . .	8
4.4. Network provides proactive control . . . . .	9
5. Summary and Proposal . . . . .	9
5.1. Reflect the network status more accurately . . . . .	10
5.2. Notify the reaction point as soon as possible. . . . .	10
6. Security Considerations . . . . .	11
7. IANA Considerations . . . . .	11
8. References . . . . .	11
8.1. Normative References . . . . .	11
8.2. Informative References . . . . .	11
Authors' Addresses . . . . .	15

## 1. Introduction

Fast congestion control is discussed in academic papers as well as in the different standard bodies. There is no one proposal for providing a solution that will work for all use cases leading to multiple approaches. By congestion control we refer to an end to end solution and not only to the congestion control algorithm on the sender side.

The major use case that we are looking at is congestion control for Data Centers, a controlled environment[RFC8085]. With the emerging Distributed Storage, AI/HPC (High Performance Computing), Machine Learning, etc., modern datacenter applications demand high throughput(40Gbps and above) with ultra-low latency of less than 10 microsecond per hop from the network, with low CPU overhead. For the end to end the latency should be less than 50usec, this value is based on DCQCN [DCQCN] The high link speed (>40Gb/s) in Data Centers (DC) are making network transfers complete faster and in fewer RTTs. Network traffic in a data center is often a mix of short and long

flows, where the short flows require low latencies and the long flows require high throughputs.

On IP-routed datacenter networks, RDMA is deployed using RoCEv2 [RoCEv2] protocol or iWARP [RFC5040] RoCEv2 [RoCEv2] is a straightforward extension of the RoCE protocol that involves a simple modification of the RoCE packet format. RoCEv2 packets carry an IP header which allows traversal of IP L3 Routers and a UDP header that serves as a stateless encapsulation layer for the RDMA Transport Protocol Packets over IP. For Data Centers RDMA in RoCEv2 expect a lossless fabric and this is achieved using ECN and PFC. iWARP congestion control is based on TCP congestion control (DCTCP [RFC8257])

A good congestion control for data centers should provide low latency, fast convergence and high link utilization. Since multiple applications with different requirements may run on the DC network it is important to provide fairness between different applications that may use different congestion algorithms. An important issue from the user perspective is to achieve short Flow Completion Time (FCT).

This document investigates the current congestion control proposals, and discusses future data center congestion control directions which aims to achieve high performance and collaboration.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Abbreviations

RCM - RoCEv2 Congestion Management

PFC - Priority-based Flow Control

ECN - Explicit Congestion Notification

DCQCN - Data Center Quantized Congestion Notification

AI/HPC - Artificial Intelligence/High-Performance computing

ECMP - Equal-Cost Multipath

NIC - Network Interface Card

RED - Random early detection gateways for congestion avoidance

#### 4. Alternative Congestion Management mechanisms

This section will describe alternative directions based on current work. Looking at the alternatives from the network perspective we can classify the alternatives as:

1. Based on estimation of network status: Traditional TCP, Timely.
2. Network provides limited information: DCQCN using only ECN, SCE and L4S
3. Network provides some information: HPCC.
4. Network provides proactive control: RCP (Rate Control Protocol)

Note that any research on congestion control that requires network participation will be irrelevant if we cannot find a viable deployment path where only part of the network devices support the proposed congestion control.

##### 4.1. Mechanisms based on estimation of network status

Traditional mechanisms uses packet status as the congestion signal and feedback to the sender, e.g. loss or delay, which is based on the facts that packets will drop when a buffer is full and packets will be delayed when a queue is building up. It can simply be achieved by the interactions between the sender and the receiver, without the involvement of network. It works well on the internet for a very long time, especially for best effort applications that do not have specific performance requirements.

However, these mechanism are not optimized for some data center application because the convergence time and throughput are not good enough. Mainly because endpoints estimation of network status are not accurate enough, and these mechanisms lack further information to adjust the sender behaviors.

##### 4.2. Network provides limited information

In these mechanisms, the network utilize the ECN field of IP header to provide some hints on network status. The following sections describe some typical proposals.

#### 4.2.1. ECN and DCTCP

The Internet solutions use ECN [RFC3168] for marking the state of the queues in the network device, they may use some AQM mechanism (fq\_codel [RFC8290] ), PIE [RFC8033]) in the network devices and a congestion algorithm (New Reno [RFC5681], Cubic [RFC8312] or DCTCP[RFC8257]) on the sender side to address the congestion in the network. Note that ECN is signaled earlier than packet drop but may cause earlier exit from TCP slow start.

One of the problem for TCP is that ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). [I-D.ietf-tcpm-accurate-ecn] specifies an alternative feedback scheme that provides more accurate information that can be used by DCTCP and L4S.

Traditional TCP uses ECN signal to indicate congestion experienced instead of packet loss, however, it does not provide information about the degree of the congestion. DCTCP [RFC8257] is trying to solve this issue. It estimates the fraction of bytes that encounter congestion rather than simply detecting the congestion presence. DCTCP further scales its sending rates accordingly. DCTCP is widely implemented in current data center environments.

#### 4.2.2. DCQCN

An enhancement to the congestion handling for ROCEv2 is the Congestion Control for Large-Scale RDMA Deployments [DCQCN] providing similar functionality to QCN [QCN] and DCTCP [RFC8257], it is implemented in some of the ROCEv2 NICs but is not part of the ROCEv2 specification. As such, vendors have their own implementations which make it difficult to interoperate with each other efficiently.

DCQCN tests are assuming that the Congestion Point is using RED-ECN for ECN marking and the RDMA CNP message is used by the Notification Point (the receiver) to report ECN Congestion Experienced (CE). DCQCN as presented includes parameters that should be set. It provides the parameters that were used during the specific tests using Mellanox NICs. One of the comments about DCQCN is that it is not simple to define the parameters in order to get an optimized solution. This solution is specific to ROCEv2 and addresses only the congestion control algorithm and is implemented in the NIC.

DCQCN notification is using CNP that only report that at least one packet with CE marking was received in the last 50usec; this is similar to TCP reporting. Other UDP based transports like RTP and QUIC provides information about how many packets marked with CE, ECT(0,1) were received.

#### 4.2.3. SCE - Some Congestion Experienced

[I-D.morton-taht-tsvwg-sce] ECT(1) to be an early notification of congestion on ECT(0) marked packets, which can be used by AQM algorithms and transports as an earlier signal of congestion than CE ("Congestion Experienced").

The ECN specification say that the congestion algorithm should treat CE marks the same as a drop packets. Using ECT(1) to signal SCE permits middleboxes implementing AQM to signal incipient congestion, below the threshold required to justify setting CE. Existing [RFC3168] compliant receivers MUST transparently ignore this new signal with respect to congestion control, and both existing and SCE-aware middleboxes MAY convert SCE to CE in the same circumstances as for ECT, thus ensuring backwards compatibility with ECN [RFC3168] endpoints.

This solution is using ECT(1) which was defined in ECN [RFC3168] as a one bit Nonce but this use is obsoleted in RFC8311 and SCE is using it for the SCE mark. There may be other documents trying to use this bit for example L4S use it to signal L4S support. The SCE marking are done by the AQM algorithm (RED, CODEL) and are sent back to the sender by the transport so there may be a need to add support for conveying the SCE marking to the sender (QUIC for example already has support for reporting the count of ECT(0) and ECT(1) separately). This solution is simpler than HPCC but provide less information.

[I-D.heist-tsvwg-sce-one-and-two-flow-tests] presents one and two-flow test results for the SCE reference implementation. These tests are not intended to be a comprehensive real-world evaluation of SCE, but an illustration of SCE's influence on basic TCP metrics in a controlled environment. The goal of the one-flow tests is to analyze the impact of SCE on the TCP throughput and TCP RTT of single TCP flows across a range of simulated path bandwidths and RTTs. The tests were with RENO and DCCP. Even though using SCE gave in general better results there were significant under-utilization at low bandwidths (<10Mb/sec; <25Mb/sec) and a slight increase in TCP RTT for DCTCP-SCE at 100Mbit / 160ms and a slight increase in TCP RTT for SCE RENO at high BDPs. The document does not describe the congestion algorithm that was used for DCTCP-SCE or RENO-SCE and comment that further work need to be done to understand the reason for this behavior.

The goal of the two-flow tests is to measure fairness between and among SCE and non-SCE TCP flows, through either a single queue or with fair queuing.

The initial results show that SCE enabled flows back off in the face of competition, whereas non-SCE flows fill the queue until a drop or CE mark occurs so fairness is not achieved. By changing the ramp by which SCE is marked and marking SCE when closer to drop or CE the fairness is better.

#### 4.2.4. L4S - Low Latency, Low Loss, Scalable Throughput

There are three main components to the L4S architecture [I-D.ietf-tsvwg-l4s-arch]

1. Network: L4S traffic needs to be isolated from the queuing latency of Classic traffic. However, the two should be able to freely share a common pool of capacity. This is because there is no way to predict how many flows at any one time might use each service and capacity in access networks is too scarce to partition into two. The Dual Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] was developed as a minimal complexity solution to this problem. The two queues appear to be separated by a 'semi-permeable' membrane that partitions latency but not bandwidth. Per-flow queuing such as in [RFC8290] could be used but it partitions both latency and bandwidth between every end-to-end flow. So it is rather overkill, which brings disadvantages, not least that large number of queues are needed when two are sufficient.
2. Protocol: A host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. [I-D.ietf-tsvwg-ecn-l4s-id] considers various alternative identifiers, and concludes that all alternatives involve compromises, but the ECT(1) and CE codepoints of the ECN field represent a workable solution.
3. Host: Scalable congestion controls already exist. They solve the scaling problem with TCP that was first pointed out in [RFC3649]. The one used most widely (in controlled environments) is Data Center TCP (DCTCP [RFC8257]). Although DCTCP as-is 'works' well over the public Internet, most implementations lack certain safety features that will be necessary once it is used outside controlled environments like data centers. A similar scalable congestion control will also need to be transplanted into protocols other than TCP (QUIC, SCTP, RTP/RTCP, RMCAT, etc.) Indeed, between the present document being drafted and published, the following scalable congestion controls were implemented: TCP Prague, QUIC Prague and an L4S variant of the RMCAT SCReAM controller [RFC8298].

Using Dual Queue provides better fairness between DCTCP and Reno/Cubic . This is less relevant to Data Centers where the competing streams may use DCQN and DCTCP.

#### 4.3. Network provides more information

The new-generation high-speed cloud network congestion control protocol HPCC (High Precision Congestion Control) [HPCC], aiming to achieve the ultimate performance and high stability of the high-speed cloud network at the same time. HPCC has been presented at ACM SIGCOMM 2019.

The key design choice of HPCC is to rely on switches to provide fine-grained load information, such as queue size and accumulated tx/rx traffic to compute precise flow rates. This has two major benefits: (i) HPCC can quickly converge to proper flow rates to highly utilize bandwidth while avoiding congestion; and (ii) HPCC can consistently maintain a close-to-zero queue for low latency.

HPCC is a sender-driven CC framework. Each packet a sender sends will be acknowledged by the receiver. During the propagation of the packet from the sender to the receiver, each switch along the path leverages the INT feature of its switching ASIC to insert some meta-data that reports the current load of the packet's egress port, including timestamp (ts), queue length (qLen), transmitted bytes (txBytes), and the link bandwidth capacity (B). When the receiver gets the packet, it copies all the meta-data recorded by the switches to the ACK message it sends back to the sender. The sender decides how to adjust its flow rate each time it receives an ACK with network load information.

Current IETF activity in IOAM [I-D.ietf-ippm-ioam-data] provides a standard mechanism for inserting metadata by the switches in the middle. IOAM can provides an optional method for sending the metadata feedback by the network to the endpoints on congestion status. But to using IOAM, the following points should be considered:

1. Is the current IOAM data fields sufficient for congestion control.
2. The encapsulation of IOAM in data center for congestion control.
3. The feedback format for sender driven congestion control.

The HPCC framework requires each node in the middle to add information about its state to the forward going packet until it reaches the receiver who will send the acknowledgment. We can think



of others modes like having the nodes in the middle updating the status information based on its available resources. This solution requires support for INT or IOAM, both protocols need to specify the packet format with the INT/IOAM extension. The HPCC document specify how to implement it for ROCEv2 while for IOAM there are some drafts in IPPM WG describing how to implement it for different transports and layer 2 packets.

The conclusion from the trials done were that HPCC can be a next-generation CC for high-speed networks to achieve ultra-low latency, high bandwidth, and stability simultaneously. HPCC achieves fast convergence, small queues, and fairness by leveraging precise load information from INT.

Similar mechanism is defined in Quick Start for TCP and IP[RFC4782]. There is a difference with the starting rate. While HPCC starts at maximum line speed [RFC4782] starts at a rate as specified in the Quick-Start request message. The Quick Start is specified for TCP, if other transport (UDP) is used there is a need to specify how the receiver send the Quick-Start response message.

#### 4.4. Network provides proactive control

The typical algorithm in this category is RCP (Rate Control Protocol) [RCP]. In the basic RCP algorithm, a router maintains a single rate,  $R(t)$ , for every link. The router "stamps"  $R(t)$  on every passing packet (unless it already carries a slower value). The receiver sends the value back to the sender, thus informing it about the slowest (or bottleneck) rate along the path. In this way, the sender quickly finds out the rate it should be using (without the need for Slow-Start). The router updates  $R(t)$  approximately once per roundtrip time, and strives to emulate Processor Sharing among flows. The biggest plus of RCP is the short flow completion times under a wide range of network and traffic characteristics.

The downside of RCP is that RCP involves the routers in congestion control, so it needs help from the infrastructure. Although they are simple, it does have per-packet computations. Another downside is that although the RCP algorithm strives to keep the buffer occupancy low most times, there are no guarantees of buffers not overflowing or of a zero packet loss.

#### 5. Summary and Proposal

Congestion control is all about how to utilize the network resource in a better and reasonably way under different network conditions. Senders are the reaction points that consume network resource, and network nodes are the congestion points. Ideally, reaction points

should react as soon as possible when network statuses change. To achieve that, there are two directions:

#### 5.1. Reflect the network status more accurately

In order to provide more information than just ECN CE marking there is a need to standardize a mechanism for the network device to provide such information and for the receiver to send more information to the sender. The network device should not insert any new fields to the IP packet but should be able to modify the value of fields in the packets sent from the data sender.

The network device will update the metadata in the forward going packet to provide more information than a single CE mark or SCE like solution.

The receiver will analyze the metadata and report back to the sender. Different from the Internet, data center network can benefit more from having more accurate information to achieve better congestion control. And this means network and hosts must collaborate together to achieve it.

Issues to be addressed:

- o How to add the metadata to the forward stream (IOAM is a valid option since we are interested in a single DC domain). The encapsulations for both IPv4 and IPv6 should be considered.
- o Negotiation of the capabilities of different nodes.
- o The format of the network information feedback to the sender in the case of sender-driven mechanisms.
- o The semantics of the message (notification or proactive)
- o Investigation of the extra load on the network device for adding the metadata.

#### 5.2. Notify the reaction point as soon as possible.

In this direction, it is worth to investigate if it's possible for the middle nodes to notify the sender directly (like IOAM Postcards) on network conditions, but such a method is challenging in terms of addressing security issues and the first concern will be that this can serve as a tool for DOS attack. But other ways, for example, carry the information in the reverse traffic would be an alternative as long as reverse traffic exists.

Issues to be addressed:

- o How to deal with multiple congestion points?
- o How to identify support by the sender and receiver for this mode and support legacy systems (same as previous mode).
- o How to authenticate the validity of the data.
- o Hardware implications

## 6. Security Considerations

TBD

## 7. IANA Considerations

No IANA action

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

### 8.2. Informative References

- [CongestionManagment] "Understanding RoCEv2 Congestion Management", 12 2018, <<https://community.mellanox.com/s/article/understanding-rocev2-congestion-management>>.
- [DCQCN] Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M. H., and M. Zhang, "Congestion control for large-scale RDMA deployments. In ACM SIGCOMM Computer Communication Review, Vol. 45. ACM, 523-536.", 8 2015, <<https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p523.pdf>>.

- [HPCC] Li, Y., Miao, R., Liur, H. H., Zhuang, Y., Feng, F., Tang, L., Cao, Z., Zhang, M., Kelly, F., Alizadeh, M., and M. Yu, "HPCC: High Precision Congestion Control", 8 2019, <<https://liyuliang001.github.io/publications/hpcc.pdf>>.
- [I-D.heist-tsvwg-sce-one-and-two-flow-tests]  
Heist, P., Grimes, R., and J. Morton, "Some Congestion Experienced One and Two-Flow Tests", draft-heist-tsvwg-sce-one-and-two-flow-tests-00 (work in progress), July 2019.
- [I-D.herbert-ipv4-eh]  
Herbert, T., "IPv4 Extension Headers and Flow Label", draft-herbert-ipv4-eh-01 (work in progress), May 2019.
- [I-D.ietf-ippm-ioam-data]  
Brockners, F., Bhandari, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mizrahi, T., Mozes, D., Lapukhov, P., Chang, R., daniel.bernier@bell.ca, d., and J. Lemon, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-07 (work in progress), September 2019.
- [I-D.ietf-quic-transport]  
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-23 (work in progress), September 2019.
- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-09 (work in progress), July 2019.
- [I-D.ietf-tsvwg-aqm-dualq-coupled]  
Schepper, K., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-10 (work in progress), July 2019.
- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-07 (work in progress), July 2019.

- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-04 (work in progress), July 2019.
- [I-D.morton-taht-tsvwg-sce]  
Morton, J. and D. Taht, "The Some Congestion Experienced ECN Codepoint", draft-morton-taht-tsvwg-sce-00 (work in progress), March 2019.
- [IEEE.802.1QBB\_2011]  
IEEE, "IEEE Standard for Local and metropolitan area networks--Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks--Amendment 17: Priority-based Flow Control", IEEE 802.1Qbb-2011, DOI 10.1109/ieeestd.2011.6032693, September 2011, <<http://ieeexplore.ieee.org/servlet/opac?punumber=6032691>>.
- [QCN]  
Alizadeh, M., Atikoglu, B., Kabbani, A., Lakshmikantha, A., Pan, R., Prabhakar, B., and M. Seaman, "Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization", 9 2008, <<https://web.stanford.edu/~balaji/papers/QCN.pdf>>.
- [RCP]  
Dukkipati, N., "RATE CONTROL PROTOCOL (RCP): CONGESTION CONTROL TO MAKE FLOWS COMPLETE QUICKLY", 10 2007, <<http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf>>.
- [RFC3168]  
Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3649]  
Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC4782]  
Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, DOI 10.17487/RFC4782, January 2007, <<https://www.rfc-editor.org/info/rfc4782>>.
- [RFC5040]  
Recio, R., Metzler, B., Culley, P., Hilland, J., and D. Garcia, "A Remote Direct Memory Access Protocol Specification", RFC 5040, DOI 10.17487/RFC5040, October 2007, <<https://www.rfc-editor.org/info/rfc5040>>.

- [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>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.
- [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>>.
- [RoCEv2] "Infiniband Trade Association. Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE).", <<https://cw.infinibandta.org/document/dl/7781>>.

Authors' Addresses

Roni Even  
Huawei

Email: roni.even@huawei.com

Rachel Huang  
Huawei Technologies Co., Ltd.

Email: rachel.huang@huawei.com