CCWG
Internet-Draft
Obsoletes: 8298 (if approved)
Intended status: Experimental
Expires: 8 January 2026

I. Johansson
M. Westerlund
M. K端hlewind
Ericsson

7 July 2025

Self-Clocked Rate Adaptation for Multimedia
draft-johansson-ccwg-rfc8298bis-screamv2-04

Abstract

This memo describes Self-Clocked Rate Adaptation for Multimedia version 2 (SCReAMv2), an update to SCReAM congestion control for media streams such as RTP [RFC3550]. SCReAMv2 includes several algorithm simplifications and adds support for L4S. The algorithm supports handling of multiple media streams, typical use cases are streaming for remote control, AR and 3D VR googles. This specification obsoletes RFC 8298.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-johansson-ccwg-rfc8298bis-screamv2/.

Discussion of this document takes place on the Congestion Control Working Group (ccwg) Working Group mailing list (mailto:ccwg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/ccwg/. Subscribe at https://www.ietf.org/mailman/listinfo/ccwg/.

Source for this draft and an issue tracker can be found at https://github.com/gloinul/draft-johansson-ccwg-scream-bis.

Status of This Memo

Table of Contents

1. Introduction

This memo describes Self-Clocked Rate Adaptation for Multimedia version 2 (SCReAMv2). This specification replaces the previous experimental version [RFC8298] of SCReAM with SCReAMv2. There are many and fairly significant changes to the original SCReAM algorithm as desribed in section Section 1.1.

Both SCReAM and SCReAMv2 estimates the forward queue delay in the same way as Low Extra Delay Background Transport (LEDBAT) [RFC6817]. However, while SCReAM is based on the self-clocking principle of TCP, SCReAMv2 is not entirely self-clocked as it augments self-clocking with pacing and a minimum send rate.

Further, SCReAMv2 can take advantage of Explicit Congestion Notification (ECN) [RFC3168] and Low Latency Low Loss and Scalable throughput (L4S) [RFC9330] in cases where ECN or L4S is supported by the network and the hosts. However, ECN or L4S is not required for the basic congestion control functionality in SCReAMv2.

1.1. Updates Compared to SCReAM (version 1)

The algorithm in this memo differs considerably compared to the previous version of SCReAM in [RFC8298]. The main differences are:

* L4S support added. The L4S algoritm has many similarities with the DCTCP and Prague congestion control but has a few extra modifications to make it work well with periodic sources such as video.

* The delay based congestion control is changed to implement a pseudo-L4S approach, this simplifies the delay based congestion control.

* The fast increase mode is removed. The reference window additive increase is replaced with an adaptive multiplicative increase to enhance convergence speed.

* The algorithm is more rate based than self-clocked:

  - The calculated congestion window is mainly used to calculate proper media bitrates. Bytes in flight is however allowed to exceeed the reference window. Therefore, The term reference window is used instead of congestion window, as the reference window does not set an absolute limit on the bytes in flight.

  - The self-clocking now acts more like an emergency break as bytes in flight can exceed the reference window only to a certain degree. The rationale is to be able to transmit large video frames and avoid that they are unnecessarily queued up on the sender side, but still prevent a large network queue.

* The media bitrate calculation is dramatically changed and simplified. In practive it is manifested with a relatively simple relation between the reference window and RTT.

* Additional compensation is added to make SCReAMv2 handle cases such as large changing frame sizes.

Algorithm changes in draft version -02 were:

* Slow down reference window growth when close to the last known maximum value is disabled and when L4S is active. This makes SCReAM adhere more closely to two marked packets per RTT at steady state.

* Reference window decrease and increase reduced by up to 50% when ref_wnd/mss is small. This reduces rate oscillations.

* Target bitrate down adjustment when ref_wnd/mss is small is modified to avoid that the data unit queue grows excessively in certain low bitrate cases.

* Timing set to multiples of RTTs instead of seconds.

Draft version -03 is major editorial pass including removal of some outdated or background information and reorganisation of several sections:

* Much shorter abstract and introduction focusing on what's new in SCReAMv2

* Removal of Section 1.1. on "Wireless (LTE and 5G) Access Properties" and Section 1.2. on "Why is it a self-clocked algorithm?"

* New Section on "Updates compared to SCReAM (version 1)" in introduction based on old Section on "Algorithm Changes"

* Section Section 1.3 updated and shortened

* Overview Section Section 3 revised; now also including the overview figure and the basic algorithms

* Own section on "Constants and variables" removed; instead all variables are now listed in the respective sections that explain the code

* New Section on "Sender Side State" explaining some basic variables

* Pseudo code and the corresponding explanations in Section Section 4.2 on "Network Congestion Control" moved into the respective subsections in section Section 4.2.1 on "Congestion Detection"

* Separate section on "Sender Transmission Control" introduced

* Section "Lost Data Unit Detection" merged into Section Section 4.2.1.1

* Section "Stream Prioritization" removed

* Section on "Competing Flows Compensation" moved into Section Section 4.2.1 on "Congestion Detection"

1.2. Requirements on the Media and Feedback Protocol

SCReAM was originally designed to with with RTP + RTCP where [RFC8888] was used as recommended feedback. RTP offers unique packet indication with the sequence number and [RFC8888] offers timestamps of received packets and the status of the ECN bits.

SCReAM is however not limited to RTP as long as some requirements are fulfilled :

* Media data is split in data units that when encapsulated in IP packets fit in the network MTU.

* Each data unit can be uniquely identified.

* Data units can be queued up in a packet queue before transmission.

* Feedback can indicate reception time for each data units, or a group of data units.

* Feedback can indicate packets that are ECN-CE marked. Unique ECN bits indication for each packet is not necessary. An ECN-CE counter similar to what is defined in [RFC9000] is sufficient.

1.3. Comparison with LEDBAT and TFWC in TCP

The core SCReAM algorithm, which is still similar in SCReAMv2, has similarities to the concepts of self-clocking used in TCP-friendly window-based congestion control [TFWC] and follows the packet conservation principle. The packet conservation principle is described as a key factor behind the protection of networks from congestion [Packet-conservation].

The reference window decrease is determined in a way similar to LEDBAT [RFC6817]. However, the window increase is not based on delay estimates but uses both a linear increase and multiplicate increase function depending on the time since the last congestion event and introduces use of inflection points in the reference window increase calculation to achieve reduced delay jitter. Further, other than LEDABT which is a scavenger congestion control mostly designed for low priority background traffic, SCReAM adjusts the qdelay target to compete with other loss-based congestion-controlled flows.

SCReAMv2 adds a new reference window validation technique, as the reference window is used as a basis for the target bitrate calculation. For that reason, various actions are taken to avoid that the reference window grows too much beyond the bytes in flight. Additional contraints are applied when in congested state and when the maximum target bitrate is reached.

The SCReAM/SCReAMv2 congestion control method uses techniques similar to LEDBAT [RFC6817] to measure the qdelay. As is the case with LEDBAT, it is not necessary to use synchronized clocks in the sender and receiver in order to compute the qdelay. However, it is necessary that they use the same clock frequency, or that the clock frequency at the receiver can be inferred reliably by the sender. Failure to meet this requirement leads to malfunction in the SCReAM/SCReAMv2 congestion control algorithm due to incorrect estimation of the network queue delay. Use of [RFC8888] as feedback ensures that the same time base is used in sender and receiver.

2. Requirements Language

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. Overview of SCReAMv2 Algorithm

SCReAMv2 consists of three main parts: network congestion control, sender transmission control, and media rate control. All of these parts reside at the sender side while the receiver is assumpted to provide acknowledgements of received data units and indication of ECN-CE marking, either as an accumulated bytes counter, or per individual data unit.

The sender implements media rate control and an data unit queue for each media type or source, where data units containing encoded media frames are temporarily stored for transmission. Figure 1 shows the details when a single media source (or stream) is used. Scheduling and priotization of mulitiple streams is not covered in this document. However, if multiple flows are sent, each data unit queue can be served based on some defined priority or simply in a round-robin fashion. Alternatively, a similar approach as coupled congestion control [RFC6365] can be applied.

```
   +--------------------------------------------------------+
   |                       Media encoder                    |
   +--------------------------------------------------------+
          ^                                       |
          |                                    Data unit
    target_bitrate                                |
          |                                       V
          |                                 +-----------+
   +------------+                           |           |
   |    Media   |                           |   Queue   |
   |    Rate    |---------------+           |   Data    |
   |   Control  |               |           |   Units   |
   +------------+               |           |           |
          ^                target_bitrate   +-----------+
          |                     |                  |
       ref_wnd                  |               Data unit
         RTT                    |                  |
          |                     |                  v
   +------------+               |          +--------------+
   |  Network   |               +---------->|    Sender    |
   | Congestion |     ref_wnd              | Transmission |
   |  Control   |-------------------------->|   Control    |
   +------------+     Bytes in flight      +--------------+
          ^                                        |
          |                                        |
   Congestion Feedback                          Data unit
     Bytes in flight                               |
          |                                        v
   +--------------------------------------------------------+
   |                         UDP socket                     |
   +--------------------------------------------------------+
```

Figure 1: Sender Functional View

Media frames are encoded and forwarded to the data unit queue in Figure 1. The data units are sent by the sender transmission controller from the data unit queue.

The sender transmission controller (in case of multiple flows a transmission scheduler) sends the data units to the UDP socket. The sender transmission controller limits the sending rate so that the number of bytes in flight is less than the reference window albeit with a slack to avoid that packets are unnecessarily delayed in the data unit queue. A pacing rate is calculated based on the target bitrate provided by the media rate controller.

Feedback about the received bytes as well as metadata to estimate the congestion level or queuing delay are provided to the network congestion controller. The network congestion controller calculated reference window and provides it togteher with the bytes in flight to the sender transmission control.

The reference window and the estimated RTT is further provided to the media rate control to compute the appropriate target bitrate. The target bitrate is updated whenever the reference window is updated. Additional parameters are also communicated to make the rate control more stable when the congestion window is very small or when L4S is not active.

3.1. Network Congestion Control

The network congestion control sets reference window (ref_wnd) which puts an upper limit on how many bytes can be in flight, i.e., transmitted but not yet acknowledged. The reference window is however not an absolute limit as slack is given to efficiently transmit temporary larger media objects, such as video frames. This means that the algoritm prefers to build up a queue in the network rather than on the sender side. Additional congestion that this causes will reflect back and cause a reduction of the reference window.

Reference window is reduced if congestion is detected. Similar as for LEDBAT the reference window is reduced either by a fixed fraction in case of packet loss or Classic ECN marking, or if the estimated queue delay exceeds a given threshold depending on how much the delay exceeds the threshold. SCReAMv2 reduces the reference window in proportion to the fraction of marked packets if L4S is used (scalable congestion control).

```
ref_wnd = BETA_LOSS * (BETA_ECN|l4s_alpha) * qtarget_alpha * ref_wnd
```

After a congestion event the reference window seeks to increase by one segment per RTT until a certain number of RTT elapses. After this initial congestion avoidance phase the refrence window increases multiplicativly where the increase factor is adjusted relative to a previous max value and the time elapsed since last congestion event. This enables a faster convergence to a higher link speed.

```
ref_wnd = ref_wnd + increment
```

3.2. Sender Transmission Control

The sender transmission control limits sending rate based on the relation of the estimated link throughput (bytes in flight) and the reference window.

```
send_wnd = ref_wnd * REF_WND_OVERHEAD * frame_size - bytes_in_flight
```

The respective sending rate is achived by applying packet pacing: Even if the send window allows for the transmission of a number of packets, these packets are not transmitted immediately; rather, they are transmitted in intervals given by the packet size and the estimated link throughput. Packets are generally paced at a higher rate than the target bitrate, this makes it possible to transmit occasionally larger video frames in a timely manner. Further, this mitigates issues with ACK compression that can cause increased jitter and/or packet loss in the media traffic.

3.3. Media Rate Control

The media rate control calculates the media rate based on the reference window and RTT.

```
target_bitrate = 8 * ref_wnd / s_rtt
```

The media rate needs to ramp up quickly enough to get a fair share of the system resources when link throughput increases. Further, the reaction to reduced throughput must be prompt in order to avoid getting too much data queued in the data unit queue(s) in the sender.

For the case that multiple streams are enabled, the media rate among the streams is distrubuted according to relative priorities.

In cases where the sender’s frame queues increase rapidly, such as in the case of a Radio Access Type (RAT) handover, the SCReAMv2 sender MAY implement additional actions, such as discarding of encoded media frames or frame skipping in order to ensure that the data unit queues are drained quickly. Frame skipping results in the frame rate being temporarily reduced. Which method to use is a design choice and is outside the scope of this algorithm description.

4. Detailed Description of SCReAMv2 Sender Algorithm

This section describes the sender-side algorithm in more detail. It is split between the network congestion control, sender transmission control, and media rate control.

4.1. Sender Side State

The sender needs to maintain sending state and state about the received feedback, as explained in the following subsections, as well as the following configuration state:

* l4s_active (false): Indicates that L4S is enabled and data units are indeed marked.

4.1.1. Status Update When Sending Data

SCReAMv2 is a window-based and byte-oriented congestion control protocol, where the number of bytes transmitted is inferred from the size of the transmitted data units. Thus, a list of transmitted data units and their respective transmission times (wall-clock time) MUST be kept for further calculation. Further the following variables are needed:

* data_unit_size (0): Size [byte] of the last transmitted data unit.

* bytes_in_flight: The number of bytes in flight is computed as the sum of the sizes of the data units ranging from the data unit most recently transmitted, down to but not including the acknowledged data unit with the highest sequence number.

bytes_in_flight can be also seen as the difference between the highest transmitted byte sequence number and the highest acknowledged byte sequence number. As an example: If a data unit with sequence number SN is transmitted and the last acknowledgement indicates SN-5 as the highest received sequence number, then bytes_in_flight is computed as the sum of the size of data units with sequence number SN-4, SN-3, SN-2, SN-1, and SN. It does not matter if, for instance, the data unit with sequence number SN-3 was lost -- the size of data unit with sequence number SN-3 will still be considered in the computation of bytes_in_flight.

* bytes_in_flight_ratio (0.0): Ratio between the bytes_in_flight and the reference window ref_wnd. This value should be computed at the beginning of the ACK processing prior to updating the highest received sequence number acked.

* ref_wnd_ratio (0.0): Ratio between MSS and ref_wnd capped to not exceed 1.0 (min(1.0, MSS / ref_wnd)).

* max_bytes_in_flight (0): The maximum number of bytes in flight in the current round trip [byte].

* max_bytes_in_flight_prev (0): The maximum number of bytes in flight in previous round trip [byte].

As bytes_in_flight can spike when congestion occurs, using the maximum of max_bytes_in_flight and max_bytes_in_flight_prev makes it more likely that an uncongested bytes_in_flight is used.

4.1.2. Status Update on Receiving Feedback

The feedback from the receiver is assumed to consist of the following elements:

* The wall-clock timestamp corresponding to the received data unit with the highest sequence number.

* data_units_acked: A list of received data units' sequence numbers.

* data_units_acked_ce: An indication if data units are ECN-CE marked. The ECN status can be either per data unit or an accumulated count of ECN-CE marked data units.

* bytes_newly_acked (0): Number of bytes newly ACKed, reset to 0 when congestion window is updated [byte].

* bytes_newly_acked_ce (0): Number of bytes newly ACKed and CE marked, reset to 0 when reference window is updated [byte].

bytes_newly_acked is incremented with a value corresponding to how much the highest sequence number has increased since the last feedback. As an example: If the previous acknowledgement indicated the highest sequence number N and the new acknowledgement indicated N+3, then bytes_newly_acked is incremented by a value equal to the sum of the sizes of data units with sequence number N+1, N+2, and N+3. Data units that are lost are also included, which means that even though, e.g., data unit N+2 was lost, its size is still included in the update of bytes_newly_acked. The bytes_newly_acked_ce is, similar to bytes_newly_acked, a counter of bytes newly acked with the extra condition that they are ECN-CE marked. The bytes_newly_acked and bytes_newly_acked_ce are reset to zero after a ref_wnd update.

4.2. Network Congestion Control

This section explains the network congestion control, which calcultes the reference window. The reference window gives an upper limit to the number of bytes in flight.

4.2.1. Congestion Detection: Delay, Data Unit Loss and ECN-CE

Congestion is detected based on three different indicators:

* Lost data units detected,

* ECN-CE marked data units detected either for classic ECN or L4S,

* Estimated queue delay exceeds a threshold.

A congestion event occurs if any of the above indicators are true AND it is at least min(VIRTUAL_RTT,s_rtt) since the last congestion event. This ensures that the reference window is reduced at most once per smoothed RTT.

4.2.1.1. Detecting Lost Data Units

The reference window back-off due to loss events is deliberately a bit less than is the case with TCP Reno, for example. TCP is generally used to transmit whole files; the file is then like a source with an infinite bitrate until the whole file has been transmitted. SCReAMv2, on the other hand, has a source which rate is limited to a value close to the available transmit rate and often below that value; the effect is that SCReAMv2 has less opportunity to grab free capacity than a TCP-based file transfer. To compensate for this, it is RECOMMENDED to let SCReAMv2 reduce the reference window less than what is the case with TCP when loss events occur.

Lost data unit detection is based on the received sequence number list. A reordering window SHOULD be applied to prevent data unit reordering from triggering loss events. The reordering window is specified as a time unit, similar to the ideas behind Recent ACKnowledgement (RACK) [RFC8985]. The computation of the reordering window is made possible by means of a lost flag in the list of transmitted data units. This flag is set if the received sequence number list indicates that the given data unit is missing. If later feedback indicates that a previously lost marked data unit was indeed received, then the reordering window is updated to reflect the reordering delay. The reordering window is given by the difference in time between the event that the data unit was marked as lost and the event that it was indicated as successfully received. Loss is detected if a given data unit is not acknowledged within a time window (indicated by the reordering window) after an data unit with a higher sequence number was acknowledged.

4.2.1.2. Receiving ECN-CE with classic ECN

In classic ECN mode the ref_wnd is scaled by a fixed value (BETA_ECN).

The reference window back-off due to an ECN event MAY be smaller than if a loss event occurs. This is in line with the idea outlined in [RFC8511] to enable ECN marking thresholds lower than the corresponding data unit drop thresholds.

4.2.1.3. Receiving ECN-CE for L4S

The ref_wnd is scaled down in proportion to the fraction of marked data units per RTT. The scale down proportion is given by l4s_alpha, which is an EWMA filtered version of the fraction of marked data units per RTT. This is inline with how DCTCP works [RFC8257]. Additional methods are applied to make the reference window reduction reasonably stable, especially when the reference window is only a few MSS. In addition, because SCReAMv2 can quite often be source limited, additional steps are taken to restore the reference window to a proper value after a long period without congestion.

l4s_alpha is calculated based in number of data units delivered (and marked) the following way:

```
data_units_delivered_this_rtt += data_units_acked
data_units_marked_this_rtt += data_units_acked_ce
# l4s_alpha is updated at least every 10ms
if (now - last_update_l4s_alpha_time >= min(0.01,s_rtt)
  # l4s_alpha is calculated from data_units marked istf bytes marked
  fraction_marked_t = data_units_marked_this_rtt/
                      data_units_delivered_this_rtt

  l4s_alpha = L4S_AVG_G*fraction_marked_t + (1.0-L4S_AVG_G)*l4S_alpha

  last_update_l4s_alpha_time = now
  data_units_delivered_this_rtt = 0
  data_units_marked_this_rtt = 0
  last_fraction_marked = fraction_marked_t
end
```

This makes calculation of L4S alpha more accurate at very low bitrates, given that the tail data unit in e.g a video frame is often smaller than MSS.

The following variables are used:

* l4s_alpha (0.0): Average fraction of marked data units per RTT.

* last_update_l4s_alpha_time (0): Last time l4s_alpha was updated [s].

* data_units_delivered_this_rtt (0): Counter for delivered data units.

* data_units_marked_this_rtt (0): Counter delivered and ECN-CE marked data units.

* last_fraction_marked (0.0): fraction marked data units in last update

The following constant is used

* L4S_AVG_G (1/16): Exponentially Weighted Moving Average (EWMA) factor for l4s_alpha

4.2.1.4. Detecting Increased Queue Delay

SCReAMv2 implements a delay-based congestion control approach where it mimics L4S congestion marking when the averaged queue delay exceeds a target threshold. This threshold is set to qdelay_target/2 and the congestion backoff factor (l4s_alpha_v) increases linearly from 0 to 100% as qdelay_avg goes from qdelay_target/2 to qdelay_target. The averaged qdelay (qdelay_avg) is used to avoid that the SCReAMv2 congestion control over-reacts to scheduling jitter, sudden delay spikes due to e.g. handover or link layer retransmissions. Furthermore, the delay based congestion control is inactivated when it is reasonably certain that L4S is active, i.e. L4S is enabled and congested nodes apply L4S marking of data units. This reduces negative effects of clockdrift, that the delay based control can introduce, whenever possible.

qdelay_avg is updated with a slow attack, fast decay EWMA filter the following way:

```
if (now - last_update_qdelay_avg_time >= s_rtt)
  if (qdelay < qdelay_avg)
    qdelay_avg = qdelay
  else
    qdelay_avg = QDELAY_AVG_G*qdelay + (1.0-QDELAY_AVG_G)*qdelay_avg
  end
  last_update_qdelay_avg_time = now
end
```

The following variables are used:

* qdelay: When the sender receives feedback, the qdelay is calculated as outlined in [RFC6817]. A qdelay sample is obtained for each received acknowledgement.

* last_update_qdelay_avg_time (0): Last time qdelay_avg was updated [s].

* s_rtt (0.0): Smoothed RTT [s], computed with a similar method to that described in [RFC6298].

The following constant is used:

* QDELAY_AVG_G (1/4): Exponentially Weighted Moving Average (EWMA) factor for qdelay_avg

4.2.1.4.1. Competing Flows Compensation

It is likely that a flow will have to share congested bottlenecks with other flows that use a more aggressive congestion control algorithm (for example, large FTP flows using loss-based congestion control). The worst condition occurs when the bottleneck queues are of tail-drop type with a large buffer size. SCReAMv2 takes care of such situations by adjusting the qdelay_target when loss-based flows are detected, as shown in the pseudocode below.

```
adjust_qdelay_target(qdelay)
  qdelay_norm_t = qdelay / QDELAY_TARGET_LOW
  update_qdelay_norm_history(qdelay_norm_t)
  # Compute variance
  qdelay_norm_var_t = VARIANCE(qdelay_norm_history(200))
  # Compensation for competing traffic
  # Compute average
  qdelay_norm_avg_t = AVERAGE(qdelay_norm_history(50))
  # Compute upper limit to target delay
  new_target_t = qdelay_norm_avg_t + sqrt(qdelay_norm_var_t)
  new_target_t *= QDELAY_TARGET_LO
  if (loss_event_rate > 0.002)
    # Data unit losses detected
    qdelay_target = 1.5 * new_target_t
  else
    if (qdelay_norm_var_t < 0.2)
      # Reasonably safe to set target qdelay
      qdelay_target = new_target_t
    else
      # Check if target delay can be reduced; this helps prevent
      # the target delay from being locked to high values forever
      if (new_target_t < QDELAY_TARGET_LO)
        # Decrease target delay quickly, as measured queuing
        # delay is lower than target
        qdelay_target = max(qdelay_target * 0.5, new_target_t)
      else
        # Decrease target delay slowly
        qdelay_target *= 0.9
      end
    end
  end

  # Apply limits
  qdelay_target = min(QDELAY_TARGET_HI, qdelay_target)
  qdelay_target = max(QDELAY_TARGET_LO, qdelay_target)
```

The follwoing variable is used:

* loss_event_rate (0.0): The estimated fraction of RTTs with lost data units detected.

Two temporary variables are calculated. qdelay_norm_avg_t is the long-term average queue delay, qdelay_norm_var_t is the long-term variance of the queue delay. A high qdelay_norm_var_t indicates that the queue delay changes; this can be an indication that bottleneck bandwidth is reduced or that a competing flow has just entered. Thus, it indicates that it is not safe to adjust the queue delay target.

A low qdelay_norm_var_t indicates that the queue delay is relatively stable. The reason could be that the queue delay is low, but it could also be that a competing flow is causing the bottleneck to reach the point that data unit losses start to occur, in which case the queue delay will stay relatively high for a longer time.

The queue delay target is allowed to be increased if either the loss event rate is above a given threshold or qdelay_norm_var_t is low. Both these conditions indicate that a competing flow may be present. In all other cases, the queue delay target is decreased.

The function that adjusts the qdelay_target is simple and could produce false positives and false negatives. The case that self-inflicted congestion by the SCReAMv2 algorithm may be falsely interpreted as the presence of competing loss-based FTP flows is a false positive. The opposite case -- where the algorithm fails to detect the presence of a competing FTP flow -- is a false negative.

Extensive simulations have shown that the algorithm performs well in LTE and 5G test cases and that it also performs well in simple bandwidth-limited bottleneck test cases with competing FTP flows. However, the potential failure of the algorithm cannot be completely ruled out. A false positive (i.e., when self-inflicted congestion is mistakenly identified as competing flows) is especially problematic when it leads to increasing the target queue delay, which can cause the end-to-end delay to increase dramatically.

If it is deemed unlikely that competing flows occur over the same bottleneck, the algorithm described in this section MAY be turned off. One such case is QoS-enabled bearers in 3GPP-based access such as LTE and 5G. However, when sending over the Internet, often the network conditions are not known for sure, so in general it is not possible to make safe assumptions on how a network is used and whether or not competing flows share the same bottleneck. Therefore, turning this algorithm off must be considered with caution, as it can lead to basically zero throughput if competing with loss-based traffic.

4.2.2. Reference Window Update

The reference window update contains two parts. One that reduces the reference window when congestion events (listed above) occur, and one part that continously increases the reference window.

The following variables are defined:

* ref_wnd (MIN_REF_WND): Reference window [byte].

* ref_wnd_i (1): Reference window inflection point [byte].

* qdelay_target (QDELAY_TARGET_LO): qdelay target [s], a variable qdelay target is introduced to manage cases where a fixed qdelay target would otherwise starve the data flow under such circumstances (e.g., FTP competes for the bandwidth over the same bottleneck). The qdelay target is allowed to vary between QDELAY_TARGET_LO and QDELAY_TARGET_HI.

* last_congestion_detected_time (0): Last time congestion event occured [s].

* last_ref_wnd_i_update_time (0): Last time ref_wnd_i was updated [s].

Further the following constants are used (the RECOMMENDED values, within parentheses "()", for the constants are deduced from experiments):

* QDELAY_TARGET_LO (0.06): Target value for the minimum qdelay [s].

* QDELAY_TARGET_HI (0.4): Target value for the maximum qdelay [s]. This parameter provides an upper limit to how much the target qdelay (qdelay_target) can be increased in order to cope with competing loss-based flows. However, the target qdelay does not have to be initialized to this high value, as it would increase end-to-end delay and also make the rate control and congestion control loops sluggish.

* MIN_REF_WND (3000): Minimum reference window [byte].

* BYTES_IN_FLIGHT_HEAD_ROOM (2.0): Extra headroom for bytes in flight.

* BETA_LOSS (0.7): ref_wnd scale factor due to loss event.

* BETA_ECN (0.8): ref_wnd scale factor due to ECN event.

* MSS (1000): Maximum segment size = Max data unit size [byte].

* REF_WND_OVERHEAD (5.0): Indicates how much bytes in flight is allowed to exceed ref_wnd.

* POST_CONGESTION_DELAY_RTT (100): Determines how many RTTs after a congestion event the reference window growth should be cautious.

* MUL_INCREASE_FACTOR (0.02): Determines how much (as a fraction of ref_wnd) that the ref_wnd can increase per RTT.

* IS_L4S (false): Congestion control operates in L4S mode.

* VIRTUAL_RTT (0.025): Virtual RTT [s]. This mimics Prague's RTT fairness such that flows with RTT below VIRTUAL_RTT should get a roughly equal share over an L4S path.

4.2.2.1. Reference Window Reduction

```
# Compute scaling factor for reference window adjustment
# when close to the last known max value before congestion
scl_t = (ref_wnd-ref_wnd_i) / ref_wnd_i
scl_t *= 8
scl_t = scl_t * scl_t
scl_t = max(0.1, min(1.0, scl_t))

# The reference window is updated at least every VIRTUAL_RTT
if (now - last_congestion_detected_time >= min(VIRTUAL_RTT,s_rtt)
  if (loss detected)
    is_loss_t = true
  end
  if (data units marked)
    is_ce_t = true
  end
  if (qdelay > qdelay_target/2)
    # It is expected that l4s_alpha is below a given value,
    l4_alpha_lim_t = 2 / target_bitrate * MSS * 8 / s_rtt
    if (l4s_alpha < l4_alpha_lim_t || !l4s_active)
      # L4S does not seem to be active
      l4s_alpha_v_t = min(1.0, max(0.0,
         (qdelay_avg - qdelay_target / 2) /
         (qdelay_target / 2)));
      is_virtual_ce_t = true
    end
  end
end

if (is_loss_t || is_ce_t || is_virtual_ce_t)
  if (now - last_ref_wnd_i_update_time > 10*s_rtt)
    # Update ref_wnd_i, no more often than every 10 RTTs
    # Additional median filtering over more congestion epochs
    # may improve accuracy of ref_wnd_i
    last_ref_wnd_i_update_time = now
    ref_wnd_i = ref_wnd
  end
end


# Either loss, ECN mark or increased qdelay is detected
```

```
if (is_loss_t)
  # Loss is detected
  ref_wnd = ref_wnd * BETA_LOSS
end
if (is_ce_t)
  # ECN-CE detected
  if (IS_L4S)
    # L4S mode
    backoff_t = l4s_alpha / 2

    # Increase stability for very small ref_wnd
    backOff_t *= max(0.5, 1.0 - ref_wnd_ratio)

    # Scale down backoff if close to the last known max reference window
    # This is complemented with a scale down of the reference window increase
    # Don’t scale down back off if queue delay is large
    if (queue_qelay < queue_delay_target * 0.25)
        backOff *= max(0.25, sclI)

    if (now - last_congestion_detected_time >
        100*max(VIRTUAL_RTT,s_rtt))
      # A long time (>100 RTTs) since last congested because
      # link throughput exceeds max video bitrate.
      # There is a certain risk that ref_wnd has increased way above
      # bytes in flight, so we reduce it here to get it better on
      # track and thus the congestion episode is shortened
      ref_wnd = min(ref_wnd, max_bytes_in_flight_prev)

      # Also, we back off a little extra if needed
      # because alpha is quite likely very low
      # This can in some cases be an over-reaction
      # but as this function should kick in relatively seldom
      # it should not be to too big concern
      backoff_t = max(backoff_t, 0.25)

      # In addition, bump up l4sAlpha to a more credible value
      # This may over react but it is better than
      # excessive queue delay
      l4sAlpha = 0.25
    end
    ref_wnd = (1.0 - backoff_t) * ref_wnd
  else
    # Classic ECN mode
    ref_wnd = ref_wnd * BETA_ECN
  end
end
if (is_virtual_ce_t)
  backoff_t = l4s_alpha_v_t / 2
```

```
  ref_wnd = (1.0 - backoff_t) * ref_wnd
end
ref_wnd = max(MIN_REF_WND, ref_wnd)

if (is_loss_t || is_ce_t || is_virtual_ce_t)
  last_congestion_detected_time = now
end
```

4.2.2.2. Reference Window Increase

```
  # Delay factor for multiplicative reference window increase
  # after congestion

  post_congestion_scale_t = max(0.0, min(1.0,
   (now - last_congestion_detected_time) /
    (POST_CONGESTION_DELAY_RTTS * max(VIRTUAL_RTT, s_rtt))))

  # Scale factor for ref_wnd update
  ref_wnd_scale_factor_t = 1.0 + (MUL_INCREASE_FACTOR  * ref_wnd) / MSS)


  # Calculate bytes acked that are not CE marked
  # For the case that only accumulated number of CE marked packets is
  # reported by the feedback, it is necessary to make an approximation
  # of bytes_newly_acked_ce based on average data unit size.
  bytes_newly_acked_minus_ce_t = bytes_newly_acked-
                                 bytes_newly_acked_ce

  increment_t = bytes_newly_acked_minus_ce_t*ref_wnd_ratio

  # Reduce increment for small RTTs
  tmp_t = min(1.0, s_rtt / VIRTUAL_RTT)
  increment_t *= tmp_t * tmp_t

  # Apply limit to reference window growth when close to last
  # known max value before congestion
  if (is_l4s_active)
     increment_t *= max(0.25,scl_t)
  else
     increment_t *= scl_t
  end

  # Limit on CWND growth speed further for small CWND
  # This is complemented with a corresponding restriction on CWND
  # reduction
  increment_t *= max(0.5,1.0-ref_wnd_ratio)

  # Scale up increment with multiplicative increase
```

```
# Limit multiplicative increase when congestion occured
# recently and when reference window is close to the last
# known max value
float tmp_t = ref_wnd_scale_factor_t
if (tmp_t > 1.0)
  tmp_t = 1.0 + (tmp_t - 1.0) * post_congestion_scale_t * scl_t;
end
increment *= tmp_t

# Increase ref_wnd only if bytes in flight is large enough
# Quite a lot of slack is allowed here to avoid that bitrate
# locks to low values.
# Increase is inhibited if max target bitrate is reached.
max_allowed_t = MSS + max(max_bytes_in_flight,
  max_bytes_in_flight_prev) * BYTES_IN_FLIGHT_HEAD_ROOM
int ref_wnd_t = ref_wnd + increment_t
if (ref_wnd_t <= max_allowed_t && target_bitrate < TARGET_BITRATE_MAX)
  ref_wnd = ref_wnd_t
end
```

The ref_wnd_scale_factor_t scales the reference window increase. The ref_wnd_scale_factor_t is increased with larger ref_wnd to allow for a multiplicative increase and thus a faster convergence when link capacity increases.

The multiplicative increase is restricted directly after a congestion event and the restriction is gradually relaxed as the time since last congested increased. The restriction makes the reference window growth to be no faster than additive increase when congestion continusly occurs. For L4S operation this means that the SCReAMv2 algorithm will adhere to the 2 marked data units per RTT equilibrium at steady state congestion, with the exception of the case below.

The reference window increase is restricted to values as small as 0.1MSS/RTT when the reference window is close to the last known max value (ref_wnd_i). This increases stability and reduces periodic overshoot. This restriction is applied in full only for small reference windows when in L4S operation.

It is particularly important that the reference window reflects the transmitted bitrate especially in L4S mode operation. An inflated ref_wnd takes extra RTTs to bring down to a correct value upon congestion and thus causes unnecessary queue buildup. At the same time the reference window must be allowed to be large enough to avoid that the SCReAMv2 algorithm begins to limit itself, given that the target bitrate is calculated based on the ref_wnd. Two mechanisms are used to manage this:

* Restore correct value of ref_wnd upon congestion. This is done if is a prolonged time since the link was congested. A typical example is that SCReAMv2 has been rate limited, i.e the target bitrate has reached the TARGET_BITRATE_MAX.

* Limit ref_wnd when the target_bitrate has reached TARGET_BITRATE_MAX. The ref_wnd is restricted based on a history of the last max_bytes_in_flight values. See [SCReAM-CPP-implementation] for details.

The two mechanisms complement one another.

4.3. Sender Transmission Control

The Sender Transmission control calculates of send window at the sender. Data units are transmitted if allowed by the relation between the number of bytes in flight and the reference window. This is controlled by the send window:

* send_wnd (0): Upper limit to how many bytes can currently be transmitted. Updated when ref_wnd is updated and when data unit is transmitted [byte].

4.3.1. Send Window Calculation

The basic design principle behind data unit transmission in SCReAM was to allow transmission only if the number of bytes in flight is less than the congestion window. There are, however, two reasons why this strict rule will not work optimally:

* Bitrate variations: Media sources such as video encoders generally produce frames whose size always vary to a larger or smaller extent. The data unit queue absorbs the natural variations in frame sizes. However, the data unit queue should be as short as possible to prevent the end-to-end delay from increasing. A strict ’send only when bytes in flight is less than the reference window’ rule can cause the data unit queue to grow simply because the send window is limited. The consequence is that the reference window will not increase, or will increase very slowly, because the reference window is only allowed to increase when there is a sufficient amount of data in flight. The final effect is that the media bitrate increases very slowly or not at all.

* Reverse (feedback) path congestion: Especially in transport over buffer-bloated networks, the one-way delay in the reverse direction can jump due to congestion. The effect is that the acknowledgements are delayed, and the self-clocking is temporarily halted, even though the forward path is not congested. The REF_WND_OVERHEAD allows for some degree of reverse path congestion as the bytes in flight is allowed to exceed ref_wnd.

In SCReAMv2, the send window is given by the relation between the adjusted reference window and the amount of bytes in flight according to the pseudocode below. The multiplication of ref_wnd with REF_WND_OVERHEAD and rel_framesize_high has the effect that bytes in flight is 'around' the ref_wnd rather than limited by the ref_wnd when the link is congested. The implementation allows the data unit queue to be small even when the frame sizes vary and thus increased e2e delay can be avoided.

```
send_wnd = ref_wnd * REF_WND_OVERHEAD * rel_framesize_high -
           bytes_in_flight
```

The send window is updated whenever an data unit is transmitted or an feedback messaged is received.

4.3.2. Calculate Frame Size

The variable rel_framesize_high is based on calculation of the high percentile of the frame sizes. The calculation is based on a histogram of the frame sizes relative to the expected frame size given the target bitrate and frame period.

* rel_framesize_high (1.0): High percentile of frame size, normalized by nominal frame size for the given target bitrate

* frame_period (0.02): The frame period [s].

* frame_size: the frame size of the last encoded frame

The calculation of rel_framesize_high is done for every new video frame and is outlined roughly with the pseudo code below:

```
tmp_t = frame_size / (target_bitrate * frame_period / 8)

if (tmp_t > 1.0)
  # Insert sample into histogram
  insert_into_histogram(tmp_t)
  # Get high percentile
  rel_framesize_high = get_histogram_high_percentile()
end
```

A 75%-ile is used in [SCReAM-CPP-implementation], the histogram can be made leaky such that old samples are gradually forgotten.

4.3.3. Packet Pacing

Packet pacing is used in order to mitigate coalescing, i.e., when packets are transmitted in bursts, with the risks of increased jitter and potentially increased packet loss. Packet pacing is also recommended to be used with L4S and also mitigates possible issues with queue overflow due to key-frame generation in video coders.

* pace_bitrate (1e6): Data unit pacing rate [bps].

* t_pace (1e-6): Pacing interval between data units [s].

The following constants are used by the packet pacing:

* RATE_PACE_MIN (50000): Minimum pacing rate in [bps].

* PACKET_PACING_HEADROOM (1.5): Extra head room for packet pacing.

The time interval between consecutive data unit transmissions is greater than or equal to t_pace, where t_pace is given by the equations below:

```
pace_bitrate = max(RATE_PACE_MIN, target_bitrate) *
               PACKET_PACING_HEADROOM
t_pace = data_unit_size * 8 / pace_bitrate
```

data_unit_size is the size of the last transmitted data unit. RATE_PACE_MIN is the minimum pacing rate.

4.4. Media Rate Control

The media rate control algorithm is executed whenever the reference window is updated and calculates the target bitrate:

* target_bitrate (0): Media target bitrate [bps].

The following constants are used by the media rate control:

* BYTES_IN_FLIGHT_LIMIT

* BYTES_IN_FLIGHT_LIMIT_COMPENSATION

* PACKET_OVERHEAD (20) : Estimated packetization overhead [byte]

* TARGET_BITRATE_MIN: Minimum target bitrate in [bps] (bits per second).

* TARGET_BITRATE_MAX: Maximum target bitrate in [bps].

The target bitrate is essentiatlly based on the reference window ref_wnd and the (smoothed) RTT s_rtt according to

target_bitrate = 8 * ref_wnd / s_rtt

The role of the media rate control is to strike a reasonable balance between a low amount of queuing in the data unit queue(s) and a sufficient amount of data to send in order to keep the data path busy. Because the reference window is updated based on loss, ECN-CE and delay, so does the target rate also update.

The code above however needs some modifications to work fine in a number of scenarios

* L4S is inactive, i.e L4S is either not enabled or congested bottlenecks do not L4S mark data units

* ref_wnd is very small, just a few MSS or smaller

The complete pseudo code for adjustment of the target bitrate is shown below

```
tmp_t = 1.0

# Limit bitrate if bytes in flight exceeds is close to or
# exceeds ref_wnd. This helps to avoid large rate fluctiations and
# variations in RTT.
if (queue_delay / queue_delay_target > 0.5 && bytes_in_flight_ratio > BYTES_IN_FLIGHT_LIM
IT)
  tmp_t /= min(BYTES_IN_FLIGHT_LIMIT_COMPENSATION,
    bytesInFlightRatio / BYTES_IN_FLIGHT_LIMIT)
end

# Scale down rate slighty when the reference window is very
# small compared to MSS
tmp_t *= 1.0 - min(0.2, max(0.0, ref_wnd_ratio - 0.1))

# Additional compensation for packetization overhead,
# important when MSS is small
tmp_t_ *= mss/(mss + PACKET_OVERHEAD)

# Calculate target bitrate and limit to min and max allowed
# values
target_bitrate = tmp_t * 8 * ref_wnd / s_rtt
target_bitrate = min(TARGET_BITRATE_MAX,
  max(TARGET_BITRATE_MIN,target_bitrate))
```

4.4.1. Handling of systematic errors in video coders

Some video encoders are prone to systematically generate an output bitrate that is systematically larger or smaller than the target bitrate. SCReAMv2 can handle some deviation inherently but for larger devation it becomes necessary to compensate for this. The algorithm for this is detailed in [SCReAM-CPP-implementation].

ToDo: A future draft version will describe this in more detail as it has been fully integrated into SCReAMv2.

5. Receiver Requirements on Feedback Intensity

The simple task of the receiver is to feed back acknowledgements with with time stamp and ECN bits indication for received data units to the sender. Upon reception of each data unit, the receiver MUST maintain enough information to send the aforementioned values to the sender via an RTCP transport- layer feedback message. The frequency of the feedback message depends on the available RTCP bandwidth. The requirements on the feedback elements and the feedback interval are described below.

SCReAMv2 benefits from relatively frequent feedback. It is RECOMMENDED that a SCReAMv2 implementation follows the guidelines below.

The feedback interval depends on the media bitrate. At low bitrates, it is sufficient with a feedback every frame; while at high bitrates, a feedback interval of roughly 5ms is preferred. At very high bitrates, even shorter feedback intervals MAY be needed in order to keep the self-clocking in SCReAMv2 working well. One indication that feedback is too sparse is that the SCReAMv2 implementation cannot reach high bitrates, even in uncongested links. More frequent feedback might solve this issue.

The numbers above can be formulated as a feedback interval function that can be useful for the computation of the desired RTCP bandwidth. The following equation expresses the feedback rate:

```
# Assume 100 byte feedback packets
rate_fb = 0.02 * [average received rate] / (100.0 * 8.0);
rate_fb = min(1000, max(10, rate_fb))

# Calculate feedback intervals
fb_int = 1.0/rate_fb
```

Feedback should also forcibly be transmitted in any of these cases:

* More than N data units received since last feedback has been transmitted

* A data unit with marker bit set or other last data unit for media frame is received

The transmission interval is not critical. So, in the case of multi-stream handling between two hosts, the feedback for two or more synchronization sources (SSRCs) can be bundled to save UDP/IP overhead. However, the final realized feedback interval SHOULD NOT exceed 2*fb_int in such cases, meaning that a scheduled feedback transmission event should not be delayed more than fb_int.

SCReAMv2 works with AVPF regular mode; immediate or early mode is not required by SCReAMv2 but can nonetheless be useful for RTCP messages not directly related to SCReAMv2, such as those specified in [RFC4585]. It is RECOMMENDED to use reduced-size RTCP [RFC5506], where regular full compound RTCP transmission is controlled by trr-int as described in [RFC4585].

While the guidelines above are somewhat RTCP specific, similar principles apply to for instance QUIC.

6. Discussion

This section covers a few discussion points.

* Clock drift: SCReAM/SCReAMv2 can suffer from the same issues with clock drift as is the case with LEDBAT [RFC6817]. However, Appendix A.2 in [RFC6817] describes ways to mitigate issues with clock drift. A clockdrift compensation method is also implemented in [SCReAM-CPP-implementation]. Furthermore, the SCReAM implementation resets base delay history when it is determined that clock drift becomes too large. This is achieved by reducing the target bitrate for a few RTTs.

* REF_WND_OVERHEAD is by default quite high. The intention is to avoid that packets are queued up on the sender side in cases when feedback is delayed or when the media encoder produces very large frames. This is beneficial for cases where link capacity is very high or when congested queues have high statistical multiplexing. It is however recommended to reduce this value in case the media encoder reacts slowly to a reduced target bitrate, because an excessive queue build-up may otherwise occur and the better option may then be to queue up packets on the sender side.

* Clock skipping: The sender or receiver clock can occasionally skip. Handling of this is implemented in [SCReAM-CPP-implementation].

* The target bitrate given by SCReAMv2 is the bitrate including the data unit and Forward Error Correction (FEC) overhead. The media encoder SHOULD take this overhead into account when the media bitrate is set. This means that the media coder bitrate SHOULD be computed as: media_rate = target_bitrate - data_unit_plus_fec_overhead_bitrate It is not necessary to make a 100% perfect compensation for the overhead, as the SCReAM algorithm will inherently compensate for moderate errors. Under-compensating for the overhead has the effect of increasing jitter, while overcompensating will cause the bottleneck link to become underutilized.

* The link utilization with SCReAMv2 can be lower than 100%. There are several possible reasons to this:

  - Large variations in frame sizes: Large variations in frame size makes SCReAMv2 push down the target_bitrate to give sufficient headroom and avoid queue buildup in the network. It is in general recommended to operate video coders in low latency mode and enable GDR (Gradual Decoding Refresh) if possible to minimize frame size variations.

- Link layer properties: Media transport in 5G in uplink typically requires to transmit a scheduling request (SR) to get persmission to transmit data. Because transmission of video is frame based, there is a high likelihood that the channel becomes idle between frames (especially with L4S), in which case a new SR/grant exchange is needed. This potentially means that uplink transmission slots are unused with a lower link utilization as a result.

* Packet pacing is recommended, it is however possible to operate SCReAMv2 with packet pacing disabled. The code in [SCReAM-CPP-implementation] implements additonal mechanisms to achieve a high link utilization when packet pacing is disabled.

* Feedback issues: RTCP feedback packets [RFC8888] can be lost, this means that the loss detection in SCReAMv2 may trigger even though packets arrive safely on the receiver side. [SCReAM-CPP-implementation] solves this by using overlapping RTCP feedback, i.e RTCP feedback is transmitted no more seldom than every 16th packet, and where each RTCP feedback spans the last 32 received packets. This however creates unnecessary overhead. [RFC3550] RR (Receiver Reports) can possibly be another solution to achieve better robustness with less overhead. QUIC [RFC9000] overcomes this issue because of inherent design.

* SCReAM has over time been evaluated in a number of different experiments, a few examples are found in [SCReAM-evaluation-L4S].

7. IANA Considerations

This document does not require any IANA actions.

8. Security Considerations

The feedback can be vulnerable to attacks similar to those that can affect TCP. It is therefore RECOMMENDED that the RTCP feedback is at least integrity protected. Furthermore, as SCReAM/SCReAMv2 is self-clocked, a malicious middlebox can drop RTCP feedback packets and thus cause the self-clocking to stall. However, this attack is mitigated by the minimum send rate maintained by SCReAM/SCReAMv2 when no feedback is received.

9. References

9.1. Normative References

[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>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <https://www.rfc-editor.org/info/rfc4585>.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, DOI 10.17487/RFC5506, April 2009, <https://www.rfc-editor.org/info/rfc5506>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP’s Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[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>.

[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>.

[RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <https://www.rfc-editor.org/info/rfc9330>.

9.2. Informative 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>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <https://www.rfc-editor.org/info/rfc7478>.

[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>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/info/rfc8511>.

[RFC8699] Islam, S., Welzl, M., and S. Gjessing, "Coupled Congestion Control for RTP Media", RFC 8699, DOI 10.17487/RFC8699, January 2020, <https://www.rfc-editor.org/info/rfc8699>.

[RFC8869] Sarker, Z., Zhu, X., and J. Fu, "Evaluation Test Cases for Interactive Real-Time Media over Wireless Networks", RFC 8869, DOI 10.17487/RFC8869, January 2021, <https://www.rfc-editor.org/info/rfc8869>.

[RFC8985] Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <https://www.rfc-editor.org/info/rfc8985>.

[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>.

[RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <https://www.rfc-editor.org/info/rfc8888>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9332] De Schepper, K., Briscoe, B., Ed., and G. White, "Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)", RFC 9332, DOI 10.17487/RFC9332, January 2023, <https://www.rfc-editor.org/info/rfc9332>.

[Packet-conservation] Jacobson, V., "Congestion Avoidance and Control", ACM SIGCOMM Computer Communication Review, DOI 10.1145/52325.52356, August 1988, <https://doi.org/10.1145/52325.52356>.

[LEDBAT-delay-impact] Ros, D. and M. Welzl, "Assessing LEDBAT’s Delay Impact", IEEE Communications Letters, Vol. 17, No. 5,, DOI 10.1109/LCOMM.2013.040213.130137, May 2013, <http://home.ifi.uio.no/michawe/research/publications/ledbat-impact-letters.pdf>.

[QoS-3GPP] "Policy and charging control architecture", 3GPP TS 23.203, July 2017, <http://www.3gpp.org/ftp/specs/archive/23_series/23.203/>.

[SCReAM-CPP-implementation] Ericsson Research, "SCReAM - Mobile optimised congestion control algorithm", n.d., <https://github.com/EricssonResearch/scream>.

[SCReAM-evaluation-L4S] Ericsson Research, "SCReAM - evaluations with L4S", n.d., <https://github.com/EricssonResearch/scream/blob/master/L4S-Results.pdf?raw=true>.

[TFWC] Choi, S. and M. Handley, "Fairer TCP-Friendly Congestion Control Protocol for Multimedia Streaming Applications", DOI 10.1145/1364654.1364717, December 2007, <http://www-dept.cs.ucl.ac.uk/staff/M.Handley/papers/tfwc-conext.pdf>.

Appendix A. Acknowledgments

Zaheduzzaman Sarker was a co-author of RFC 8298 the previous version of scream which this document was based on. We would like to thank the following people for their comments, questions, and support during the work that led to this memo: Mirja Kuehlewind.

Authors’ Addresses

Ingemar Johansson
Ericsson
Email: ingemar.s.johansson@ericsson.com

Magnus Westerlund
Ericsson
Email: magnus.westerlund@ericsson.com

Mirja K端hlewind
Ericsson
Email: mirja.kuehlewind@ericsson.com