                Self-Clocked Rate Adaptation for Multimedia

Abstract

   This memo describes a rate adaptation algorithm for conversational
   media services such as interactive video. The solution conforms to
   the packet conservation principle and uses a hybrid loss-and-delay-
   based congestion control algorithm. The algorithm is evaluated over
   both simulated Internet bottleneck scenarios as well as in a Long
   Term Evolution (LTE) system simulator and is shown to achieve both
   low latency and high video throughput in these scenarios.

Status of This Memo

Table of Contents

1. Introduction

   Congestion in the Internet occurs when the transmitted bitrate is
   higher than the available capacity over a given transmission path.
   Applications that are deployed in the Internet have to employ
   congestion control to achieve robust performance and to avoid
   congestion collapse in the Internet. Interactive real-time
   communication imposes a lot of requirements on the transport;
   therefore, a robust, efficient rate adaptation for all access types
   is an important part of interactive real-time communications, as the
   transmission channel bandwidth can vary over time. Wireless access
   such as LTE, which is an integral part of the current Internet,
   increases the importance of rate adaptation as the channel bandwidth
   of a default LTE bearer [QoS-3GPP] can change considerably in a very
   short time frame. Thus, a rate adaptation solution for interactive
   real-time media, such as WebRTC [RFC7478], should be both quick and
   be able to operate over a large range in channel capacity. This memo
   describes Self-Clocked Rate Adaptation for Multimedia (SCReAM), a
   solution that implements congestion control for RTP streams
   [RFC3550]. While SCReAM was originally devised for WebRTC, it can
   also be used for other applications where congestion control of RTP
   streams is necessary. SCReAM is based on the self-clocking principle
   of TCP and uses techniques similar to what is used in the rate
   adaptation algorithm based on Low Extra Delay Background Transport
   (LEDBAT) [RFC6817]. SCReAM is not entirely self-clocked as it
   augments self-clocking with pacing and a minimum send rate. SCReAM
   can take advantage of Explicit Congestion Notification (ECN) in cases
   where ECN is supported by the network and the hosts. However, ECN is
   not required for the basic congestion control functionality in
   SCReAM.

1.1. Wireless (LTE) Access Properties

   [WIRELESS-TESTS] describes the complications that can be observed in
   wireless environments. Wireless access such as LTE typically cannot
   guarantee a given bandwidth; this is true especially for default
   bearers. The network throughput can vary considerably, for instance,
   in cases where the wireless terminal is moving around. Even though
   LTE can support bitrates well above 100 Mbps, there are cases when
   the available bitrate can be much lower; examples are situations with
   high network load and poor coverage. An additional complication is
   that the network throughput can drop for short time intervals (e.g.,
   at handover); these short glitches are initially very difficult to
   distinguish from more permanent reductions in throughput.

   Unlike wireline bottlenecks with large statistical multiplexing, it
   is not possible to try to maintain a given bitrate when congestion is
   detected with the hope that other flows will yield. This is because

there are generally few other flows competing for the same
bottleneck. Each user gets its own variable throughput bottleneck,
where the throughput depends on factors like channel quality, network
load, and historical throughput. The bottom line is, if the
throughput drops, the sender has no other option than to reduce the
bitrate. Once the radio scheduler has reduced the resource
allocation for a bearer, a flow (which is using RTP Media Congestion
Avoidance Techniques (RMCAT)) in that bearer aims to reduce the
sending rate quite quickly (within one RTT) in order to avoid
excessive queuing delay or packet loss.

1.2. Why is it a self-clocked algorithm?

   Self-clocked congestion control algorithms provide a benefit over
   their rate-based counterparts in that the former consists of two
   adaptation mechanisms:

   o A congestion window computation that evolves over a longer
      timescale (several RTTs) especially when the congestion window
      evolution is dictated by estimated delay (to minimize
      vulnerability to, e.g., short-term delay variations).

   o A fine-grained congestion control given by the self-clocking; it
      operates on a shorter time scale (1 RTT). The benefits of self-
      clocking are also elaborated upon in [TFWC].

   A rate-based congestion control algorithm typically adjusts the rate
   based on delay and loss. The congestion detection needs to be done
   with a certain time lag to avoid overreaction to spurious congestion
   events such as delay spikes. Despite the fact that there are two or
   more congestion indications, the outcome is that there is still only
   one mechanism to adjust the sending rate. This makes it difficult to
   reach the goals of high throughput and prompt reaction to congestion.

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 SCReAM Algorithm

   The core SCReAM algorithm 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].

   In SCReAM, the receiver of the media echoes a list of received RTP
   packets and the timestamp of the RTP packet with the highest sequence
   number back to the sender in feedback packets. The sender keeps a
   list of transmitted packets, their respective sizes, and the time
   they were transmitted. This information is used to determine the
   number of bytes that can be transmitted at any given time instant. A
   congestion window puts an upper limit on how many bytes can be in
   flight, i.e., transmitted but not yet acknowledged.

   The congestion window is determined in a way similar to LEDBAT
   [RFC6817]. LEDBAT is a congestion control algorithm that uses send
   and receive timestamps to estimate the queuing delay (from now on
   denoted "qdelay") along the transmission path. This information is
   used to adjust the congestion window. The use of LEDBAT ensures that
   the end-to-end latency is kept low. [LEDBAT-delay-impact] shows that
   LEDBAT has certain inherent issues that make it counteract its
   purpose of achieving low delay. The general problem described in the
   paper is that the base delay is offset by LEDBAT's own queue buildup.
   The big difference with using LEDBAT in the SCReAM context lies in
   the facts that the source is rate limited and that the RTP queue must
   be kept short (preferably empty). In addition, the output from a
   video encoder is rarely constant bitrate; static content (talking
   heads, for instance) gives almost zero video bitrate. This yields
   two useful properties when LEDBAT is used with SCReAM; they help to
   avoid the issues described in [LEDBAT-delay-impact]:

   1. There is always a certain probability that SCReAM is short of
       data to transmit; this means that the network queue will become
       empty every once in a while.

   2. The max video bitrate can be lower than the link capacity. If
       the max video bitrate is 5 Mbps and the capacity is 10 Mbps, then
       the network queue will become empty.

   It is sufficient that any of the two conditions above is fulfilled to
   make the base delay update properly. Furthermore,
   [LEDBAT-delay-impact] describes an issue with short-lived competing
   flows. In SCReAM, these short-lived flows will cause the self-
   clocking to slow down, thereby building up the RTP queue; in turn,
   this results in a reduced media video bitrate. Thus, SCReAM slows

the bitrate more when there are competing short-lived flows than the
traditional use of LEDBAT does. The basic functionality in the use
of LEDBAT in SCReAM is quite simple; however, there are a few steps
in order to make the concept work with conversational media:

o Congestion window validation techniques. These are similar to the
   method described in [RFC7661]. Congestion window validation
   ensures that the congestion window is limited by the actual number
   bytes in flight; this is important especially in the context of
   rate-limited sources such as video. Lack of congestion window
   validation would lead to a slow reaction to congestion as the
   congestion window does not properly reflect the congestion state
   in the network. The allowed idle period in this memo is shorter
   than in [RFC7661]; this to avoid excessive delays in the cases
   where, e.g., wireless throughput has decreased during a period
   where the output bitrate from the media coder has been low (for
   instance, due to inactivity). Furthermore, this memo allows for
   more relaxed rules for when the congestion window is allowed to
   grow; this is necessary as the variable output bitrate generally
   means that the congestion window is often underutilized.

o Fast increase mode makes the bitrate increase faster when no
   congestion is detected. It makes the media bitrate ramp up within
   5 to 10 seconds. The behavior is similar to TCP slowstart. Fast
   increase mode is exited when congestion is detected. However,
   fast increase mode can resume if the congestion level is low; this
   enables a reasonably quick rate increase in case link throughput
   increases.

o A qdelay trend is computed for earlier detection of incipient
   congestion; as a result, it reduces jitter.

o Addition of a media rate control function.

o Use of inflection points in the media rate calculation to achieve
   reduced jitter.

o Adjustment of qdelay target for better performance when competing
   with other loss-based congestion-controlled flows.

The above-mentioned features will be described in more detail in
Sections 3.1 to 3.3. The full details are described in Section 4.

```
                  +---------------------------+
                  |       Media encoder       |
                  +---------------------------+
                     ^                  |
                     |                  |(1)
                     |(3)               RTP
                     |                  V
                     |               +-----------+
        +---------+  |               |           |
        | Media   |  (2)             |   Queue   |
        | rate    |  |<------|       |           |
        | control |  |               |RTP packets|
        +---------+                  |           |
                                     +-----------+
                                          |
                                          |(4)
                                          RTP
                                          |
                                          v
        +-----------+         +--------------+
        |  Network  |   (7)   |    Sender    |
    +-->| congestion|-------->|  Transmission|
    |   |  control  |         |    Control   |
    |   +-----------+         +--------------+
    |                               |
    |-------------RTCP----------|    |(5)
        (6)                     |    RTP
                                |    v
                             +------------+
                             |    UDP     |
                             |   socket   |
                             +------------+
```
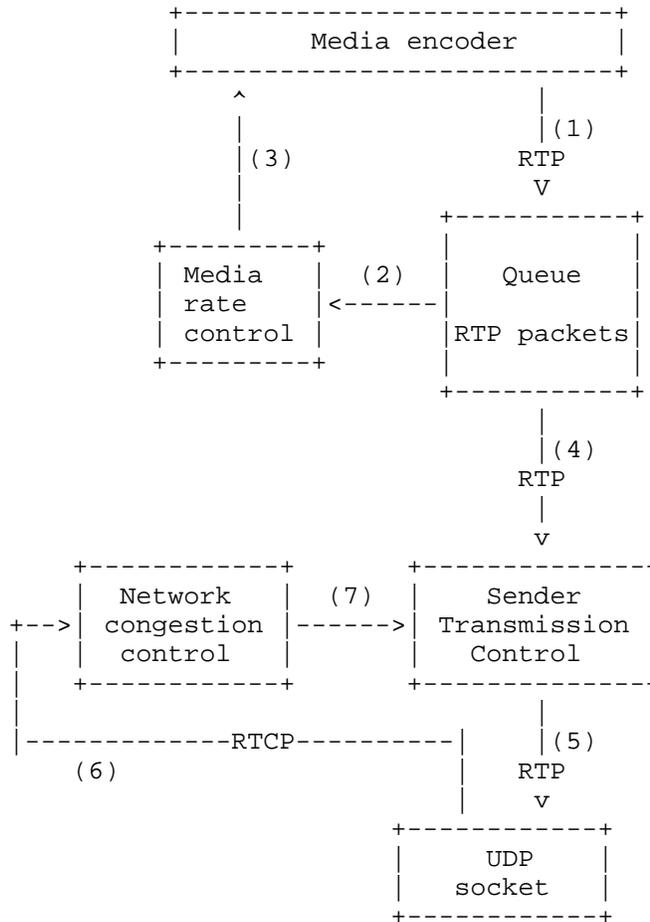
                 Figure 1: SCReAM Sender Functional View

   The SCReAM algorithm consists of three main parts: network congestion
   control, sender transmission control, and media rate control. All of
   these parts reside at the sender side. Figure 1 shows the functional
   overview of a SCReAM sender. The receiver-side algorithm is very
   simple in comparison, as it only generates feedback containing
   acknowledgements of received RTP packets and an ECN count.

3.1. Network Congestion Control

   The network congestion control sets an upper limit on how much data
   can be in the network (bytes in flight); this limit is called CWND
   (congestion window) and is used in the sender transmission control.

The SCReAM 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 congestion
control algorithm due to incorrect estimation of the network queue
delay.

The SCReAM sender calculates the congestion window based on the
feedback from the SCReAM receiver. The congestion window is allowed
to increase if the qdelay is below a predefined qdelay target;
otherwise, the congestion window decreases. The qdelay target is
typically set to 50-100 ms. This ensures that the queuing delay is
kept low. The reaction to loss or ECN events leads to an instant
reduction of CWND. Note that the source rate-limited nature of real-
time media, such as video, typically means that the queuing delay
will mostly be below the given delay target. This is contrary to the
case where large files are transmitted using LEDBAT congestion
control and the queuing delay will stay close to the delay target.

3.2. Sender Transmission Control

   The sender transmission control limits the output of data, given by
   the relation between the number of bytes in flight and the congestion
   window. Packet pacing is used to mitigate issues with ACK
   compression that MAY cause increased jitter and/or packet loss in the
   media traffic. Packet pacing limits the packet transmission rate
   given by the estimated link throughput. 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.

3.3. Media Rate Control

   The media rate control serves to adjust the media bitrate to ramp up
   quickly enough to get a fair share of the system resources when link
   throughput increases.

   The reaction to reduced throughput MUST be prompt in order to avoid
   getting too much data queued in the RTP packet queue(s) in the
   sender. The media bitrate is decreased if the RTP queue size exceeds
   a threshold.

   In cases where the sender's frame queues increase rapidly, such as in
   the case of a Radio Access Type (RAT) handover, the SCReAM sender MAY
   implement additional actions, such as discarding of encoded media

frames or frame skipping in order to ensure that the RTP 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 SCReAM

4.1. SCReAM Sender

   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.

   A SCReAM sender implements media rate control and an RTP queue for
   each media type or source, where RTP packets containing encoded media
   frames are temporarily stored for transmission. Figure 1 shows the
   details when a single media source (or stream) is used. A
   transmission scheduler (not shown in the figure) is added to support
   multiple streams. The transmission scheduler can enforce differing
   priorities between the streams and act like a coupled congestion
   controller for multiple flows. Support for multiple streams is
   implemented in [SCReAM-CPP-implementation].

   Media frames are encoded and forwarded to the RTP queue (1) in
   Figure 1. The media rate adaptation adapts to the size of the RTP
   queue (2) and provides a target rate for the media encoder (3). The
   RTP packets are picked from the RTP queue (4), for multiple flows
   from each RTP queue based on some defined priority order or simply in
   a round-robin fashion, by the sender transmission controller. The
   sender transmission controller (in case of multiple flows a
   transmission scheduler) sends the RTP packets to the UDP socket (5).
   In the general case, all media SHOULD go through the sender
   transmission controller and is limited so that the number of bytes in
   flight is less than the congestion window. RTCP packets are received
   (6) and the information about the bytes in flight and congestion
   window is exchanged between the network congestion control and the
   sender transmission control (7).

4.1.1. Constants and Parameter Values

   Constants and state variables are listed in this section. Temporary
   variables are not listed; instead, they are appended with '_t' in the
   pseudocode to indicate their local scope.

4.1.1.1. Constants

   The RECOMMENDED values, within parentheses "()", for the constants
   are deduced from experiments.

   QDELAY_TARGET_LO (0.1 s)
     Target value for the minimum qdelay.

   QDELAY_TARGET_HI (0.4 s)
     Target value for the maximum qdelay. 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.

   QDELAY_WEIGHT (0.1)
     Averaging factor for qdelay_fraction_avg.

   QDELAY_TREND_TH (0.2)
     Threshold for the detection of incipient congestion.

   MIN_CWND (3000 bytes)
     Minimum congestion window.

   MAX_BYTES_IN_FLIGHT_HEAD_ROOM (1.1)
     Headroom for the limitation of CWND.

   GAIN (1.0)
     Gain factor for congestion window adjustment.

   BETA_LOSS (0.8)
     CWND scale factor due to loss event.

   BETA_ECN (0.9)
     CWND scale factor due to ECN event.

   BETA_R (0.9)
     Scale factor for target rate due to loss event.

   MSS (1000 byte)
     Maximum segment size = Max RTP packet size.

   RATE_ADJUST_INTERVAL (0.2 s)
     Interval between media bitrate adjustments.

   TARGET_BITRATE_MIN
     Minimum target bitrate in bps (bits per second).

   TARGET_BITRATE_MAX
      Maximum target bitrate in bps.

   RAMP_UP_SPEED (200000 bps/s)
      Maximum allowed rate increase speed.

   PRE_CONGESTION_GUARD (0.0..1.0)
      Guard factor against early congestion onset. A higher value gives
      less jitter, possibly at the expense of a lower link utilization.
      This value MAY be subject to tuning depending on e.g., media coder
      characteristics. Experiments with H264 and VP8 indicate that 0.1
      is a suitable value. See [SCReAM-CPP-implementation] and
      [SCReAM-implementation-experience] for evaluation of a real
      implementation.

   TX_QUEUE_SIZE_FACTOR (0.0..2.0)
      Guard factor against RTP queue buildup. This value MAY be subject
      to tuning depending on, e.g., media coder characteristics.
      Experiments with H264 and VP8 indicate that 1.0 is a suitable
      value. See [SCReAM-CPP-implementation] and
      [SCReAM-implementation-experience] for evaluation of a real
      implementation.

   RTP_QDELAY_TH (0.02 s) RTP queue delay threshold for a target rate
      reduction.

   TARGET_RATE_SCALE_RTP_QDELAY (0.95) Scale factor for target rate
      when RTP qdelay threshold exceeds RTP_QDELAY_TH.

   QDELAY_TREND_LO (0.2) Threshold value for qdelay_trend.

   T_RESUME_FAST_INCREASE (5 s) Time span until fast increase mode can
      be resumed, given that the qdelay_trend is below QDELAY_TREND_LO.

   RATE_PACE_MIN (50000 bps) Minimum pacing rate.

4.1.1.2. State Variables

   The values within parentheses "()" indicate initial values.

   qdelay_target (QDELAY_TARGET_LO)
      qdelay target, a variable qdelay target is introduced to manage
      cases where a fixed qdelay target would otherwise starve the RMCAT
      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.

   qdelay_fraction_avg (0.0)
     Fractional qdelay filtered by the Exponentially Weighted Moving
     Average (EWMA).

   qdelay_fraction_hist[20] ({0,..,0})
     Vector of the last 20 fractional qdelay samples.

   qdelay_trend (0.0)
     qdelay trend; indicates incipient congestion.

   qdelay_trend_mem (0.0)
     Low-pass filtered version of qdelay_trend.

   qdelay_norm_hist[100] ({0,..,0})
     Vector of the last 100 normalized qdelay samples.

   in_fast_increase (true)
     True if in fast increase mode.

   cwnd (MIN_CWND)
     Congestion window.

   bytes_newly_acked (0)
     The number of bytes that was acknowledged with the last received
     acknowledgement, i.e., bytes acknowledged since the last CWND
     update.

   max_bytes_in_flight (0)
     The maximum number of bytes in flight over a sliding time window,
     i.e., transmitted but not yet acknowledged bytes.

   send_wnd (0)
     Upper limit to how many bytes can currently be transmitted.
     Updated when cwnd is updated and when RTP packet is transmitted.

   target_bitrate (0 bps)
     Media target bitrate.

   target_bitrate_last_max (1 bps)
     Inflection point of the media target bitrate, i.e., the last known
     highest target_bitrate. Used to limit bitrate increase speed close
     to the last known congestion point.

   rate_transmit (0.0 bps)
     Measured transmit bitrate.

   rate_ack (0.0 bps)
     Measured throughput based on received acknowledgements.

   rate_media (0.0 bps)
     Measured bitrate from the media encoder.

   rate_media_median (0.0 bps)
     Median value of rate_media, computed over more than 10 s.

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

   rtp_queue_size (0 bits)
     Sum of the sizes of RTP packets in queue.

   rtp_size (0 byte)
     Size of the last transmitted RTP packet.

   loss_event_rate (0.0)
     The estimated fraction of RTTs with lost packets detected.

4.1.2. Network Congestion Control

   This section explains the network congestion control, which performs
   two main functions:

   o Computation of congestion window at the sender: This gives an
      upper limit to the number of bytes in flight.

   o Calculation of send window at the sender: RTP packets are
      transmitted if allowed by the relation between the number of bytes
      in flight and the congestion window. This is controlled by the
      send window.

   SCReAM is a window-based and byte-oriented congestion control
   protocol, where the number of bytes transmitted is inferred from the
   size of the transmitted RTP packets. Thus, a list of transmitted RTP
   packets and their respective transmission times (wall-clock time)
   MUST be kept for further calculation.

   The number of bytes in flight (bytes_in_flight) is computed as the
   sum of the sizes of the RTP packets ranging from the RTP packet most
   recently transmitted, down to but not including the acknowledged
   packet with the highest sequence number. This can be translated to
   the difference between the highest transmitted byte sequence number
   and the highest acknowledged byte sequence number. As an example: If
   an RTP packet 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
   RTP packets with sequence number SN-4, SN-3, SN-2, SN-1, and SN. It

does not matter if, for instance, the packet with sequence number
SN-3 was lost -- the size of RTP packet with sequence number SN-3
will still be considered in the computation of bytes_in_flight.

Furthermore, a variable 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 RTP packets with sequence
number N+1, N+2, and N+3. Packets that are lost are also included,
which means that even though, e.g., packet N+2 was lost, its size is
still included in the update of bytes_newly_acked. The
bytes_newly_acked variable is reset to zero after a CWND update.

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

o A list of received RTP packets' sequence numbers.

o The wall-clock timestamp corresponding to the received RTP packet
   with the highest sequence number.

o The accumulated number of ECN-CE-marked packets (n_ECN). Here,
   "CE" refers to "Congestion Experienced".

When the sender receives RTCP feedback, the qdelay is calculated as
outlined in [RFC6817]. A qdelay sample is obtained for each received
acknowledgement. No smoothing of the qdelay is performed; however,
some smoothing occurs anyway because the CWND computation is a low-
pass filter function. A number of variables are updated as
illustrated by the pseudocode below; temporary variables are appended
with '_t'. As mentioned in Section 6, calculation of the proper
congestion window and media bitrate may benefit from additional
optimizations to handle very high and very low bitrates, and from
additional damping to handle periodic packet bursts. Some such
optimizations are implemented in [SCReAM-CPP-implementation], but
they do not form part of the specification of SCReAM at this time.

```
<CODE BEGINS>
update_variables(qdelay):
  qdelay_fraction_t = qdelay / qdelay_target
  # Calculate moving average
  qdelay_fraction_avg = (1 - QDELAY_WEIGHT) * qdelay_fraction_avg +
      QDELAY_WEIGHT * qdelay_fraction_t
  update_qdelay_fraction_hist(qdelay_fraction_t)
  # Compute the average of the values in qdelay_fraction_hist
  avg_t = average(qdelay_fraction_hist)
  # R is an autocorrelation function of qdelay_fraction_hist,
  #  with the mean (DC component) removed, at lag K
  # The subtraction of the scalar avg_t from
  #  qdelay_fraction_hist is performed element-wise
  a_t = R(qdelay_fraction_hist-avg_t, 1) /
        R(qdelay_fraction_hist-avg_t, 0)
  # Calculate qdelay trend
  qdelay_trend = min(1.0, max(0.0, a_t * qdelay_fraction_avg))
  # Calculate a 'peak-hold' qdelay_trend; this gives a memory
  #  of congestion in the past
  qdelay_trend_mem = max(0.99 * qdelay_trend_mem, qdelay_trend)
<CODE ENDS>
```

The qdelay fraction is sampled every 50 ms, and the last 20 samples
are stored in a vector (qdelay_fraction_hist). This vector is used
in the computation of a qdelay trend that gives a value between 0.0
and 1.0 depending on the estimated congestion level. The prediction
coefficient 'a_t' has positive values if qdelay shows an increasing
or decreasing trend; thus, an indication of congestion is obtained
before the qdelay target is reached. As a side effect, if qdelay
decreases, it's taken as a sign of congestion; however, experiments
have shown that this is beneficial, as increasing or decreasing queue
delay is an indication that the transmit rate is very close to the
path capacity.

The autocorrelation function 'R' is defined as follows. Let x be a
vector constituting N values, the biased autocorrelation function for
a given lag=k for the vector x is given by.

$$R(x,k) = \sum_{n=1}^{n=N-k} x(n) * x(n + k)$$

The prediction coefficient is further multiplied with
qdelay_fraction_avg to reduce sensitivity to increasing qdelay when
it is very small. The 50 ms sampling is a simplification that could
have the effect that the same qdelay is sampled several times;
however, this does not pose any problem, as the vector is only used
to determine if the qdelay is increasing or decreasing. The

   qdelay_trend is utilized in the media rate control to indicate
   incipient congestion and to determine when to exit from fast increase
   mode. qdelay_trend_mem is used to enforce a less aggressive rate
   increase after congestion events. The function
   update_qdelay_fraction_hist(..) removes the oldest element and adds
   the latest qdelay_fraction element to the qdelay_fraction_hist
   vector.

4.1.2.1. Reaction to Packet Loss and ECN

   A loss event is indicated if one or more RTP packets are declared
   missing. The loss detection is described in Section 4.1.2.4. Once a
   loss event is detected, further detected lost RTP packets SHOULD be
   ignored for a full smoothed round-trip time; the intention is to
   limit the congestion window decrease to at most once per round trip.

   The congestion 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. SCReAM, on the other hand, has a source whose rate is
   limited to a value close to the available transmit rate and often
   below that value; the effect is that SCReAM has less opportunity to
   grab free capacity than a TCP-based file transfer. To compensate for
   this, it is RECOMMENDED to let SCReAM reduce the congestion window
   less than what is the case with TCP when loss events occur.

   An ECN event is detected if the n_ECN counter in the feedback report
   has increased since the previous received feedback. Once an ECN
   event is detected, the n_ECN counter is ignored for a full smoothed
   round-trip time; the intention is to limit the congestion window
   decrease to at most once per round trip. The congestion 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 [ALT-BACKOFF] to enable ECN
   marking thresholds lower than the corresponding packet drop
   thresholds.

4.1.2.2. Congestion Window Update

   The update of the congestion window depends on if loss, ECN-marking,
   or neither of the two occurs. The pseudocode below describes the
   actions for each case.

```
      <CODE BEGINS>
      on congestion event(qdelay):
        # Either loss or ECN mark is detected
        in_fast_increase = false
        if (is loss)
          # Loss is detected
          cwnd = max(MIN_CWND, cwnd * BETA_LOSS)
        else
          # No loss, so it is then an ECN mark
          cwnd = max(MIN_CWND, cwnd * BETA_ECN)
        end
        adjust_qdelay_target(qdelay) #compensating for competing flows
        calculate_send_window(qdelay, qdelay_target)

      # When no congestion event
      on acknowledgement(qdelay):
        update_bytes_newly_acked()
        update_cwnd(bytes_newly_acked)
        adjust_qdelay_target(qdelay) # compensating for competing flows
        calculate_send_window(qdelay, qdelay_target)
        check_to_resume_fast_increase()
      <CODE ENDS>
```

   The methods are described in detail below.

   The congestion window update is based on qdelay, except for the
   occurrence of loss events (one or more lost RTP packets in one RTT)
   or ECN events, which were described earlier.

   Pseudocode for the update of the congestion window is found below.

```
   <CODE BEGINS>
   update_cwnd(bytes_newly_acked):
     # In fast increase mode?
     if (in_fast_increase)
       if (qdelay_trend >= QDELAY_TREND_TH)
         # Incipient congestion detected; exit fast increase mode
         in_fast_increase = false
       else
         # No congestion yet; increase cwnd if it
         #  is sufficiently used
         # Additional slack of bytes_newly_acked is
         #  added to ensure that CWND growth occurs
         #  even when feedback is sparse
         if (bytes_in_flight * 1.5 + bytes_newly_acked > cwnd)
           cwnd = cwnd + bytes_newly_acked
         end
         return
       end
     end

     # Not in fast increase mode
     # off_target calculated as with LEDBAT
     off_target_t = (qdelay_target - qdelay) / qdelay_target

     gain_t = GAIN
     # Adjust congestion window
     cwnd_delta_t =
       gain_t * off_target_t * bytes_newly_acked * MSS / cwnd
     if (off_target_t > 0 &&
         bytes_in_flight * 1.25 + bytes_newly_acked <= cwnd)
       # No cwnd increase if window is underutilized
       # Additional slack of bytes_newly_acked is
       #  added to ensure that CWND growth occurs
       #  even when feedback is sparse
       cwnd_delta_t = 0;
     end

     # Apply delta
     cwnd += cwnd_delta_t
     # limit cwnd to the maximum number of bytes in flight
     cwnd = min(cwnd, max_bytes_in_flight *
               MAX_BYTES_IN_FLIGHT_HEAD_ROOM)
     cwnd = max(cwnd, MIN_CWND)

   <CODE ENDS>
```

CWND is updated differently depending on whether or not the
congestion control is in fast increase mode, as controlled by the
variable in_fast_increase.

When in fast increase mode, the congestion window is increased with
the number of newly acknowledged bytes as long as the window is
sufficiently used. Sparse feedback can potentially limit congestion
window growth; therefore, additional slack is added, given by the
number of newly acknowledged bytes.

The congestion window growth when in_fast_increase is false is
dictated by the relation between qdelay and qdelay_target; congestion
window growth is limited if the window is not used sufficiently.

SCReAM calculates the GAIN in a similar way to what is specified in
[RFC6817]. However, [RFC6817] specifies that the CWND increase is
limited by an additional function controlled by a constant
ALLOWED_INCREASE. This additional limitation is removed in this
specification.

Further, the CWND is limited by max_bytes_in_flight and MIN_CWND.
The limitation of the congestion window by the maximum number of
bytes in flight over the last 5 seconds (max_bytes_in_flight) avoids
possible overestimation of the throughput after, for example, idle
periods. An additional MAX_BYTES_IN_FLIGHT_HEAD_ROOM provides slack
to allow for a certain amount of variability in the media coder
output rate.

4.1.2.3. Competing Flows Compensation

It is likely that a flow using the SCReAM algorithm 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. SCReAM takes care of such situations by adjusting the
qdelay_target when loss-based flows are detected, as shown in the
pseudocode below.

```
  <CODE BEGINS>
  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)
      # Packet 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)
  <CODE ENDS>
```

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 packet 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 SCReAM 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 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. 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.

4.1.2.4. Lost Packet Detection

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

4.1.2.5. Send Window Calculation

The basic design principle behind packet transmission in SCReAM is 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:

o Bitrate variations: Media sources such as video encoders generally
   produce frames whose size always vary to a larger or smaller
   extent. The RTP queue absorbs the natural variations in frame
   sizes. However, the RTP queue should be as short as possible to
   prevent the end-to-end delay from increasing. To achieve that,
   the media rate control takes the RTP queue size into account when
   the target bitrate for the media is computed. A strict 'send only
   when bytes in flight is less than the congestion window' rule can
   cause the RTP queue to grow simply because the send window is
   limited; in turn, this can cause the target bitrate to be pushed
   down. The consequence is that the congestion window will not
   increase, or will increase very slowly, because the congestion
   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.

o 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 send window is adjusted depending on qdelay, its relation to the
qdelay target, and the relation between the congestion window and the
number of bytes in flight. A strict rule is applied when qdelay is
higher than qdelay_target, to avoid further queue buildup in the
network. For cases when qdelay is lower than the qdelay_target, a
more relaxed rule is applied. This allows the bitrate to increase
quickly when no congestion is detected while still being able to
exhibit stable behavior in congested situations.

The send window is given by the relation between the adjusted
congestion window and the amount of bytes in flight according to the
pseudocode below.

```
<CODE BEGINS>
calculate_send_window(qdelay, qdelay_target)
  # send window is computed differently depending on congestion level
  if (qdelay <= qdelay_target)
    send_wnd = cwnd + MSS - bytes_in_flight
  else
    send_wnd = cwnd - bytes_in_flight
  end
<CODE ENDS>
```

The send window is updated whenever an RTP packet is transmitted or
an RTCP feedback messaged is received.

4.1.2.6. 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 also mitigates
possible issues with queue overflow due to key-frame generation in
video coders. The time interval between consecutive packet
transmissions is greater than or equal to t_pace, where t_pace is
given by the equations below :

```
<CODE BEGINS>
pace_bitrate = max (RATE_PACE_MIN, cwnd * 8 / s_rtt)
t_pace = rtp_size * 8 / pace_bitrate
<CODE ENDS>
```

rtp_size is the size of the last transmitted RTP packet, and s_rtt is
the smoothed round trip time. RATE_PACE_MIN is the minimum pacing
rate.

4.1.2.7. Resuming Fast Increase Mode

Fast increase mode can resume in order to speed up the bitrate
increase if congestion abates. The condition to resume fast increase
mode (in_fast_increase = true) is that qdelay_trend is less than
QDELAY_TREND_LO for T_RESUME_FAST_INCREASE seconds or more.

4.1.2.8. Stream Prioritization

The SCReAM algorithm makes a good distinction between network
congestion control and media rate control. This is easily extended
to many streams -- RTP packets from two or more RTP queues are
scheduled at the rate permitted by the network congestion control.

The scheduling can be done by means of a few different scheduling
regimes. For example, the method for coupled congestion control

specified in [COUPLED-CC] can be used. One implementation of SCReAM
[SCReAM-CPP-implementation] uses credit-based scheduling. In credit-
based scheduling, credit is accumulated by queues as they wait for
service and is spent while the queues are being serviced. For
instance, if one queue is allowed to transmit 1000 bytes, then a
credit of 1000 bytes is allocated to the other unscheduled queues.
This principle can be extended to weighted scheduling, where the
credit allocated to unscheduled queues depends on the relative
weights. The latter is also implemented in
[SCReAM-CPP-implementation].

4.1.3. Media Rate Control

   The media rate control algorithm is executed at regular intervals,
   indicated by RATE_ADJUSTMENT_INTERVAL, with the exception of a prompt
   reaction to loss events. The media rate control operates based on
   the size of the RTP packet send queue and observed loss events. In
   addition, qdelay_trend is also considered in the media rate control
   in order to reduce the amount of induced network jitter.

   The role of the media rate control is to strike a reasonable balance
   between a low amount of queuing in the RTP queue(s) and a sufficient
   amount of data to send in order to keep the data path busy. Setting
   the media rate control too cautiously leads to possible
   underutilization of network capacity; this can cause the flow to
   become starved out by other more opportunistic traffic. On the other
   hand, setting it too aggressively leads to increased jitter.

   The target_bitrate is adjusted depending on the congestion state.
   The target bitrate can vary between a minimum value
   (TARGET_BITRATE_MIN) and a maximum value (TARGET_BITRATE_MAX).
   TARGET_BITRATE_MIN SHOULD be set to a low enough value to prevent RTP
   packets from becoming queued up when the network throughput is
   reduced. The sender SHOULD also be equipped with a mechanism that
   discards RTP packets when the network throughput becomes very low and
   RTP packets are excessively delayed.

   For the overall bitrate adjustment, two network throughput estimates
   are computed :

   o rate_transmit: The measured transmit bitrate.

   o rate_ack: The ACKed bitrate, i.e., the volume of ACKed bits per
      second.

   Both estimates are updated every 200 ms.

The current throughput, current_rate, is computed as the maximum
value of rate_transmit and rate_ack. The rationale behind the use of
rate_ack in addition to rate_transmit is that rate_transmit is
affected also by the amount of data that is available to transmit,
thus a lack of data to transmit can be seen as reduced throughput
that can cause an unnecessary rate reduction. To overcome this
shortcoming, rate_ack is used as well. This gives a more stable
throughput estimate.

The rate change behavior depends on whether a loss or ECN event has
occurred and whether the congestion control is in fast increase mode.

```
<CODE BEGINS>
# The target_bitrate is updated at a regular interval according
# to RATE_ADJUST_INTERVAL

on loss:
   # Loss event detected
   target_bitrate = max(BETA_R * target_bitrate,
                        TARGET_BITRATE_MIN)
   exit
on ecn_mark:
   # ECN event detected
   target_bitrate = max(BETA_ECN * target_bitrate,
                        TARGET_BITRATE_MIN)
   exit

ramp_up_speed_t = min(RAMP_UP_SPEED, target_bitrate / 2.0)
scale_t = (target_bitrate - target_bitrate_last_max) /
     target_bitrate_last_max
scale_t = max(0.2, min(1.0, (scale_t * 4)^2))
# min scale_t value 0.2, as the bitrate should be allowed to
#  increase slowly. This prevents locking the rate to
#  target_bitrate_last_max
if (in_fast_increase = true)
   increment_t = ramp_up_speed_t * RATE_ADJUST_INTERVAL
   increment_t *= scale_t
   target_bitrate += increment_t
else
   current_rate_t = max(rate_transmit, rate_ack)
   # Compute a bitrate change
   delta_rate_t = current_rate_t * (1.0 - PRE_CONGESTION_GUARD *
        queue_delay_trend) - TX_QUEUE_SIZE_FACTOR * rtp_queue_size
   # Limit a positive increase if close to target_bitrate_last_max
   if (delta_rate_t > 0)
     delta_rate_t *= scale_t
     delta_rate_t =
       min(delta_rate_t, ramp_up_speed_t * RATE_ADJUST_INTERVAL)
```

```
      end
      target_bitrate += delta_rate_t
      # Force a slight reduction in bitrate if RTP queue
      #  builds up
      rtp_queue_delay_t = rtp_queue_size / current_rate_t
      if (rtp_queue_delay_t > RTP_QDELAY_TH)
        target_bitrate *= TARGET_RATE_SCALE_RTP_QDELAY
      end
  end

  rate_media_limit_t =
      max(current_rate_t, max(rate_media, rtp_rate_median))
  rate_media_limit_t *= (2.0 - qdelay_trend_mem)
  target_bitrate = min(target_bitrate, rate_media_limit_t)
  target_bitrate = min(TARGET_BITRATE_MAX,
      max(TARGET_BITRATE_MIN, target_bitrate))
<CODE ENDS>
```

In case of a loss event, the target_bitrate is updated and the rate
change procedure is exited. Otherwise, the rate change procedure
continues. The rationale behind the rate reduction due to loss is
that a congestion window reduction will take effect, and a rate
reduction proactively prevents RTP packets from being queued up when
the transmit rate decreases due to the reduced congestion window. A
similar rate reduction happens when ECN events are detected.

The rate update frequency is limited by RATE_ADJUST_INTERVAL, unless
a loss event occurs. The value is based on experimentation with
real-life limitations in video coders taken into account
[SCReAM-CPP-implementation]. A too short interval is shown to make
the rate control loop in video coders more unstable; a too long
interval makes the overall congestion control sluggish.

When in fast increase mode (in_fast_increase = true), the bitrate
increase is given by the desired ramp-up speed (RAMP_UP_SPEED). The
ramp-up speed is limited when the target bitrate is low to avoid rate
oscillation at low bottleneck bitrates. The setting of RAMP_UP_SPEED
depends on preferences. A high setting such as 1000 kbps/s makes it
possible to quickly get high-quality media; however, this is at the
expense of increased jitter, which can manifest itself as choppy
video rendering, for example.

When in_fast_increase is false, the bitrate increase is given by the
current bitrate and is also controlled by the estimated RTP queue and
the qdelay trend, thus it is sufficient that an increased congestion
level is sensed by the network congestion control to limit the
bitrate. The target_bitrate_last_max is updated when congestion is
detected.

Finally, the target_bitrate is within the defined min and max values.

The aware reader may notice the dependency on the qdelay in the computation of the target bitrate; this manifests itself in the use of the qdelay_trend. As these parameters are used also in the network congestion control, one may suspect some odd interaction between the media rate control and the network congestion control. This is in fact the case if the parameter PRE_CONGESTION_GUARD is set to a high value. The use of qdelay_trend in the media rate control is solely to reduce jitter; the dependency can be removed by setting PRE_CONGESTION_GUARD=0. The effect is a somewhat larger rate increase after congestion, at the expense of increased jitter in congested situations.

4.2. SCReAM Receiver

The simple task of the SCReAM receiver is to feed back acknowledgements of received packets and total ECN count to the SCReAM sender. In addition, the receive time of the RTP packet with the highest sequence number is echoed back. Upon reception of each RTP packet, the receiver MUST maintain enough information to send the aforementioned values to the SCReAM 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.

4.2.1. Requirements on Feedback Elements

The following feedback elements are REQUIRED for basic functionality in SCReAM.

o A list of received RTP packets. This list SHOULD be sufficiently long to cover all received RTP packets. This list can be realized with the Loss RLE (Run Length Encoding) Report Block in [RFC3611].

o A wall-clock timestamp corresponding to the received RTP packet with the highest sequence number is required in order to compute the qdelay. This can be realized by means of the Packet Receipt Times Report Block in [RFC3611]. begin_seq MUST be set to the highest received sequence number (which has possibly wrapped around); end_seq MUST be set to begin_seq+1 modulo 65536. The timestamp clock MAY be set according to [RFC3611], i.e., equal to the RTP timestamp clock. Detailed individual packet receive times are not necessary, as SCReAM does currently not describe how they can be used.

The basic feedback needed for SCReAM involves the use of the Loss RLE
Report Block and the Packet Receipt Times Report Block as shown in
Figure 2.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |V=2|P|reserved |    PT=XR=207    |              length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             SSRC                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     BT=2       | rsvd. |  T=0  |          block length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         SSRC of source                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        begin_seq          |              end_seq               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         chunk 1           |              chunk 2               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   :                            ...                                 :
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        chunk n-1          |              chunk n               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     BT=3       | rsvd. |  T=0  |          block length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         SSRC of source                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        begin_seq          |              end_seq               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Receipt time of packet begin_seq                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
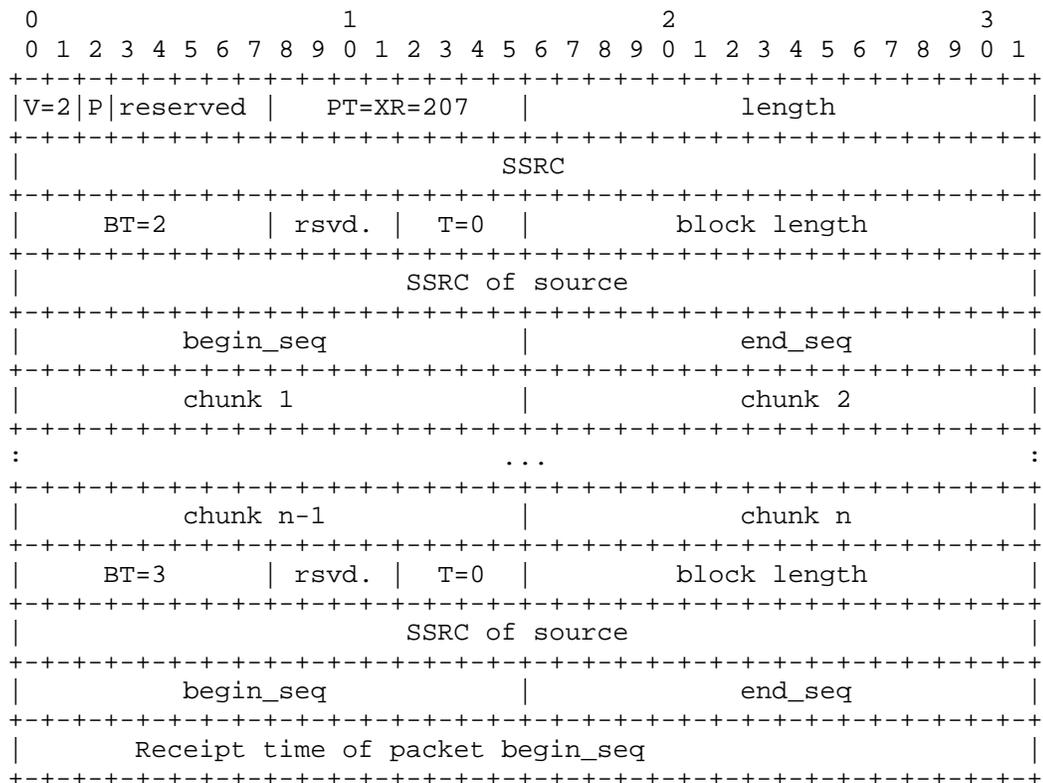
        Figure 2: Basic Feedback Message for SCReAM, Based on RFC 3611

In a typical use case, no more than four Loss RLE chunks are needed,
thus the feedback message will be 44 bytes. It is obvious from
Figure 2 that there is a lot of redundant information in the feedback
message. A more optimized feedback format, including the additional
feedback elements listed below, could reduce the feedback message
size a bit.

An additional feedback element that can improve the performance of
SCReAM is:

o Accumulated number of ECN-CE-marked packets (n_ECN). For
   instance, this can be realized with the ECN Feedback Report Format
   in [RFC6679]. The given feedback report format is slightly
   overkill, as SCReAM would do quite well with only a counter that

increments by one for each received packet with the ECN-CE
codepoint set. The more bulky format could nevertheless be useful
for, e.g., ECN black-hole detection.

4.2.2. Requirements on Feedback Intensity

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

The feedback interval depends on the media bitrate. At low bitrates,
it is sufficient with a feedback interval of 100 to 400 ms; while at
high bitrates, a feedback interval of roughly 20 ms is preferred. At
very high bitrates, even shorter feedback intervals MAY be needed in
order to keep the self-clocking in SCReAM working well. One
indication that feedback is too sparse is that the SCReAM
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:

    rate_fb = min(50, max(2.5, rate_media / 10000))

rate_media is the RTP media bitrate expressed in bps; rate_fb is the
feedback rate expressed in packets/s. Converting to feedback
interval, we get:

    fb_int = 1.0 / min(50, max(2.5, rate_media / 10000))

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.

SCReAM works with AVPF regular mode; immediate or early mode is not
required by SCReAM but can nonetheless be useful for RTCP messages
not directly related to SCReAM, 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].

5. Discussion

   This section covers a few discussion points.

   o Clock drift: SCReAM 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.

   o Support for alternate ECN semantics: This specification adopts the
      proposal in [ALT-BACKOFF] to reduce the congestion window less
      when ECN-based congestion events are detected. Future work on Low
      Loss, Low Latency for Scalable throughput (L4S) may lead to
      updates in a future document that describes SCReAM support for
      L4S.

   o A new transport-layer feedback message (as specified in RFC 4585)
      could be standardized if the use of the already existing RTCP
      extensions as described in Section 4.2 is not deemed sufficient.

   o The target bitrate given by SCReAM is the bitrate including the
      RTP 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 - rtp_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.

6. Suggested Experiments

   SCReAM has been evaluated in a number of different ways, mostly in a
   simulator. The OpenWebRTC implementation work ([OpenWebRTC] and
   [SCReAM-implementation]) involved extensive testing with artificial
   bottlenecks with varying bandwidths and using two different video
   coders (OpenH264 and VP9).

      Preferably, further experiments will be done by means of
      implementation in real clients and web browsers. RECOMMENDED
      experiments are:

      o Trials with various access technologies: EDGE/3G/4G, Wi-Fi, DSL.
         Some experiments have already been carried out with LTE access;
         see [SCReAM-CPP-implementation] and
         [SCReAM-implementation-experience].

      o Trials with different kinds of media: Audio, video, slideshow
         content. Evaluation of multi-stream handling in SCReAM.

      o Evaluation of functionality of the compensation mechanism when
         there are competing flows: Evaluate how SCReAM performs with
         competing TCP-like traffic and to what extent the compensation for
         competing flows causes self-inflicted congestion.

      o Determine proper parameters: A set of default parameters are given
         that makes SCReAM work over a reasonably large operation range.
         However, for very low or very high bitrates, it may be necessary
         to use different values for the RAMP_UP_SPEED, for instance.

      o Experimentation with further improvements to the congestion window
         and media bitrate calculation. [SCReAM-CPP-implementation]
         implements some optimizations, not described in this memo, that
         improve performance slightly. Further experiments are likely to
         lead to more optimizations of the algorithm.

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 is self-clocked, a
      malicious middlebox can drop RTCP feedback packets and thus cause the
      self-clocking in SCReAM to stall. However, this attack is mitigated
      by the minimum send rate maintained by SCReAM when no feedback is
      received.

9. References

9.1. 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>.

   [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>.

   [RFC3611] Friedman, T., Ed., Caceres, R., Ed., and A. Clark, Ed.,
              "RTP Control Protocol Extended Reports (RTCP XR)",
              RFC 3611, DOI 10.17487/RFC3611, November 2003,
              <https://www.rfc-editor.org/info/rfc3611>.

   [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>.

9.2. Informative References

   [ALT-BACKOFF]
              Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst,
              "TCP Alternative Backoff with ECN (ABE)", Work in
              Progress, draft-ietf-tcpm-alternativebackoff-ecn-04,
              November 2017.

   [COUPLED-CC]
              Islam, S., Welzl, M., and S. Gjessing, "Coupled congestion
              control for RTP media", Work in Progress, draft-ietf-
              rmcat-coupled-cc-07, September 2017.

   [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>.

   [OpenWebRTC]
              Ericsson Research, "OpenWebRTC",
              <http://www.openwebrtc.org>.

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

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

   [RACK]   Cheng, Y., Cardwell, N., and N. Dukkipati, "RACK: a time-
              based fast loss detection algorithm for TCP", Work in
              Progress, draft-ietf-tcpm-rack-02, March 2017.

   [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>.

   [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>.

   [RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating
              TCP to Support Rate-Limited Traffic", RFC 7661,
              DOI 10.17487/RFC7661, October 2015,
              <https://www.rfc-editor.org/info/rfc7661>.

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

   [SCReAM-implementation]
              Ericsson Research, "OpenWebRTC specific GStreamer
              plugins", <https://github.com/EricssonResearch/
              openwebrtc-gst-plugins>.

   [SCReAM-implementation-experience]
              Sarker, Z. and I. Johansson, "Updates on SCReAM: An
              implementation experience", November 2015,
              <https://www.ietf.org/proceedings/94/slides/
              slides-94-rmcat-8.pdf>.

   [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>.

   [WIRELESS-TESTS]
              Sarker, Z., Johansson, I., Zhu, X., Fu, J., Tan, W., and
              M. Ramalho, "Evaluation Test Cases for Interactive Real-
              Time Media over Wireless Networks", Work in Progress,
              draft-ietf-rmcat-wireless-tests-04, May 2017.

Authors' Addresses

   Ingemar Johansson
   Ericsson AB
   Laboratoriegraend 11
   Luleaa 977 53
   Sweden

   Phone: +46 730783289
   Email: ingemar.s.johansson@ericsson.com

   Zaheduzzaman Sarker
   Ericsson AB
   Laboratoriegraend 11
   Luleaa 977 53
   Sweden

   Phone: +46 761153743
   Email: zaheduzzaman.sarker@ericsson.com