RMCAT WG
Internet-Draft
Intended status: Experimental
Expires: August 11, 2016

I. Johansson
Z. Sarker
Ericsson AB

February 8, 2016

Self-Clocked Rate Adaptation for Multimedia
draft-ietf-rmcat-scream-cc-03

Abstract

This memo describes a rate adaptation algorithm for conversational media services such as 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 LTE (Long Term Evolution) system simulator and is shown to achieve both low latency and high video throughput in these scenarios.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 11, 2016.

Table of Contents

1. Introduction

Congestion in the Internet is a reality and applications that are deployed in the Internet must have congestion control schemes in place not only for the robustness of the service that it provides but also to ensure the function of the currently deployed Internet. As the interactive realtime communication imposes a great deal of requirements on the transport, a robust, efficient rate adaptation for all access types is considered as an important part of interactive realtime communications as the transmission channel bandwidth may 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 realtime media, such as WebRTC, must be both quick and be able to operate over a large span in available channel bandwidth. This memo describes a solution,named SCReAM, that is based on the self-clocking principle of TCP and uses techniques similar to what is used in a new delay based rate adaptation algorithm, LEDBAT [RFC6817].

1.1. Wireless (LTE) access properties

[I-D.ietf-rmcat-wireless-tests] describes the complications that can be observed in wireless environments. Wireless access such as LTE can typically not guarantee a given bandwidth, this is true especially for default bearers. The network throughput may vary considerably for instance in cases where the wireless terminal is moving around.

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, an RMCAT flow in that bearer needs to reduce the sending rate quite quickly (in 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 algorithm provides with a benefit over the rate based counterparts in that the former consists of two parts; the 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) and; the fine grained congestion control given by the self-clocking which operates on a shorter time scale (1 RTT). The benefits of self-clocking are also elaborated upon in [TFWC].

A rate based congestion control typically adjusts the rate based on delay and loss. The congestion detection needs to be done with a certain time lag to avoid over-reaction to spurious congestion events such as delay spikes. Despite the fact that there are two or more congestion indications, the outcome is still that there is 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. Terminology

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

3. Overview of SCReAM Algorithm

The core SCReAM algorithm has similarities to the concepts of self-clocking used in TFWC [TFWC] and follows the packet conservation principle. The packet conservation principle is described as an important 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 amount 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. This realizes the packet conservation principle. 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 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. The basic functionality is quite simple, there are however a few steps to take to make the concept work with conversational media. In a few words they are:

- Congestion window validation techniques. These are similar in action as the method described in [RFC7661]. Congestion window validation ensures that the congestion window is limited by the amount of actual bytes in flight, this is important especially in the context of rate limited sources which is the case when video is transmitted. 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 the reference, this to avoid excessive delays in the cases where e.g. wireless throughput has decreased during a period where the output bitrate has been low. 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 under-utilized.

- Fast increase for quicker bitrate increase. It makes the media bitrate ramp-up within 5 to 10 seconds. The behavior is similar to TCP slowstart. The fast increase is exited when congestion is detected. The fast increase state can however resume if the congestion level is low, this to enable a reasonably quick rate increase in case link throughput increases.

- A delay trend is computed for earlier detection of incipient congestion and as a result it reduces jitter.

- Addition of a media rate control function.

- Use of inflection points to calculate congestion window and media rate to achieve reduced jitter.

- Adjustment of delay 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 Section 3.1 to Section 3.3.

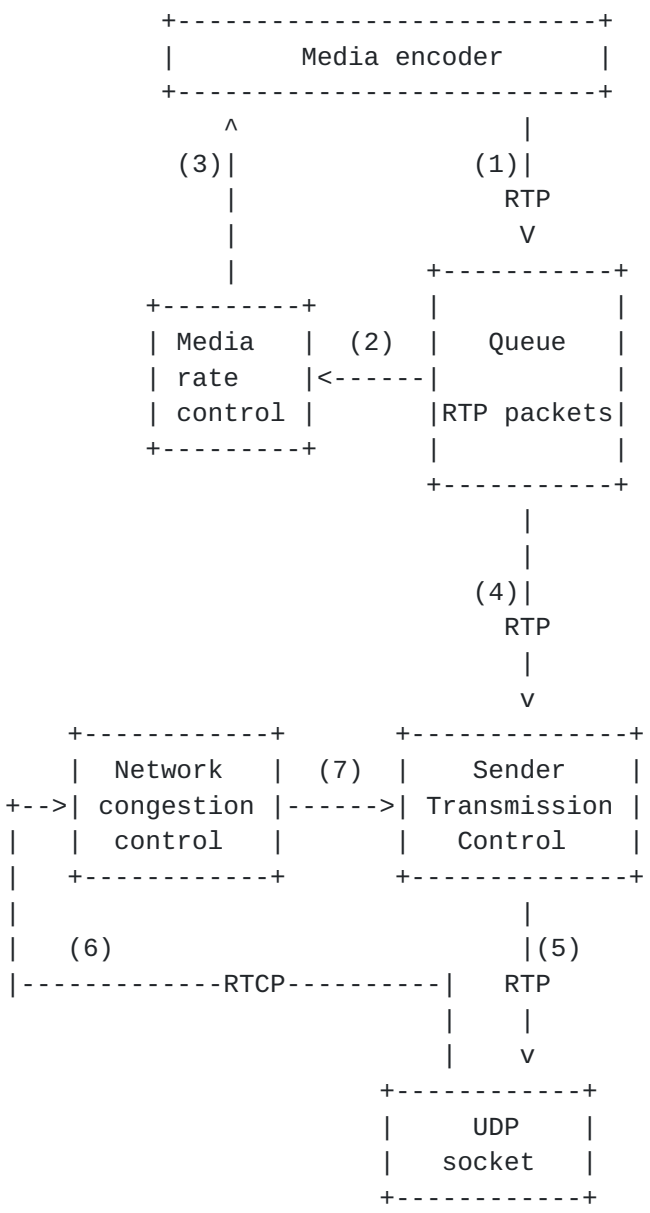

Figure 1: SCReAM sender functional view

The SCReAM algorithm constitutes mainly three parts: network congestion control, sender transmission control and media rate control. All these three parts reside at the sender side. Figure 2 shows the functional overview of a SCReAM sender. The receiver side algorithm is very simple in comparison as it only generates feedback containing acknowledgements to received RTP packets and 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 queuing delay, also termed qdelay in this memo for brevity. Similar to LEDBAT, it is not necessary to use synchronized clocks in sender and receiver in order to compute the queuing delay. It is however necessary that they use the same clock frequency, or that the clock frequency at the receiver can be inferred reliably by the sender.

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 delay target is typically set to 50-100ms. 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, in which case 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, this has the effect that even if the send window allows for the transmission of a number of packets, these packets are not transmitted immediately, but rather they are transmitted in intervals given by the packet size and the 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 up 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 frame queues increase rapidly such as the case of a RAT (Radio Access Type) handover it may be necessary to implement additional actions, such as discarding of encoded media frames or frame skipping in order to ensure that the RTP queues are drained quickly or simply that stale RTP packets are removed from the queue. Frame skipping means that the frame rate is temporarily reduced. Which method to use is a design consideration and 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 a split between the network congestion control, sender transmission control and the media rate control.

A SCReAM sender implements media rate control and a 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 (a.k.a stream) is used. Multiple media sources are also supported in the design, in that case the sender transmission control will include a transmission scheduler. The transmission scheduler can then enforce the priorities for the different streams and then act like a coupled congestion controller for multiple flows.

Media frames are encoded and forwarded to the RTP queue (1). The media rate adaptation adapts to the size of the RTP queue (2) and controls the media bitrate (3). The RTP packets are picked from the RTP queue (for multiple flows from each RTP queue based on some defined priority order or simply in a round robin fashion) (4) by the sender transmission controller. The sender transmission controller (in case of multiple flows a transmission scheduler) takes care of the transmission of RTP packets, to be written to the UDP socket (5). In the general case all media must go through the sender transmission controller and is allowed to be transmitted if the number of bytes in flight is less than the congestion window. RTCP packets are received (6) and the information about 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 pseudo code to indicate their local scope.

4.1.1.1. Constants

The recommended values for the constants are deduced from experimentals.

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

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

QDELAY_WEIGHT (0.1)
  Averaging factor for qdelay_fraction_avg.

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 (0.6)
  CWND scale factor due to loss event.

BETA_ECN (0.8)
  CWND scale factor due to ECN event.

BETA_R (0.9)
  Target rate scale factor due to loss event.

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

RATE_ADJUST_INTERVAL (0.2s)
  Interval between media bitrate adjustments.

TARGET_BITRATE_MIN
  Min target bitrate [bps].

TARGET_BITRATE_MAX
  Max target bitrate [bps].

RAMP_UP_SPEED (200000bps/s)

Maximum allowed rate increase speed.

PRE_CONGESTION_GUARD (0.0..0.2)
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 have however given that 0.1 is a suitable value.

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 have however given that 1.0 is a suitable value.

QDELAY_TREND_LO (0.2) Threshold value for qdelay_trend.

T_RESUME_FAST_INCREASE Time span until fast increase can be resumed, given that the qdelay_trend is below QDELAY_TREND_LO.

4.1.1.2. State variables

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

qdelay_fraction_avg (0.0)
EWMA filtered fractional qdelay.

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.

min_cwnd (2*MSS)
Minimum congestion window.

in_fast_increase (true)
True if in fast increase state.

cwnd (min_cwnd)
Congestion window.

cwnd_last_max (1 byte)
Congestion window inflection point, i.e. the last known highest cwnd. Used to limit cwnd increase speed close to the last known congestion point.

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.

send_wnd (0)
Upper limit to how many bytes that 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)
Media target bitrate inflection point 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 10s.

s_rtt (0.0s)
Smoothed RTT [s], computed similar to method depicted in [RFC6298]

rtp_queue_size (0 bits)
Size of RTP packets in queue.

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

4.1.2. Network congestion control

This section explains the network congestion control, it contains two main functions

- Computation of congestion window at the sender: Gives an upper limit to the number of bytes in flight i.e. how many bytes that have been transmitted but not yet acknowledged.

- 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.

Unlike TCP, SCReAM is not a byte oriented protocol, rather it is an RTP packet oriented protocol. Thus a list of transmitted RTP packets and their respective transmission times (wall-clock time) is kept for further calculation. The congestion control is however based on transmitted and acknowledged bytes.

SCReAM uses the terminology "Bytes in flight" (bytes_in_flight) which 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 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 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 is reset after a CWND update.

The feedback from the receiver is assumed to consist of the following elements. More details are found in Appendix A.4.

- A list of received RTP packets.

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

- Accumulated number of ECN-CE marked packets (n_ECN).

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 samples occur, however some smoothing occurs anyway as the computation of the CWND is in itself a low pass filter function. A number of variables are updated as illustrated by the pseudo code below.

```
  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)
    # R is an autocorrelation function of qdelay_fraction_hist
    #  at lag K
    a = R(qdelay_fraction_hist,1)/R(qdelay_fraction_hist,0)
    #calculate qdelay trend
    qdelay_trend = min(1.0,max(0.0,a*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)
```

The qdelay fraction is sampled every 50ms and the last 20 samples are stored in a vector (qdelay_fraction_hist). This vector is used in the computation of an qdelay trend that gives a value between 0.0 and 1.0 depending on the estimated congestion level. The prediction coefficient 'a' has positive values if qdelay shows an increasing trend, thus an indication of congestion is obtained before the qdelay target is reached. The autocorrelation function 'R' is defined in Appendix A.2. The prediction coefficient is further multiplied with qdelay_fraction_avg to reduce sensitivity to increasing qdelay when it is very small. The 50ms sampling is a simplification and may have the effect that the same qdelay is sampled several times, this is however not a big issue as the vector is only used for the computation of qdelay_trend. 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.

A loss event is indicated if one or more RTP packets are declared missing. The loss detection is described in Section 4.1.2.3. Once a loss event is detected, further detected lost RTP packets are ignored for a full smoothed round trip time, the intention of this is to limit the congestion window decrease to at most once per round trip. The congestion window backoff due to loss events is deliberately a bit less than is the case with e.g TCP NewReno. The reason is that TCP is generally used to transmit whole files, which can be translated to an infinite source bitrate. SCReAM on the other hand has a source which rate is limited to a value close to the available transmit rate and often below said value, the effect of this is that SCReAM has less opportunity to grab free capacity than a TCP based file transfer. To compensate for this it is necessary to let SCReAM reduce the congestion window slightly less 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 of this is to limit the congestion window decrease to at most once per round trip. The congestion window backoff due to an ECN event is deliberately smaller than if a loss event occurs. This is inline with the idea outlined in [Khademi_alternative_backoff_ECN] to enable ECN marking thresholds lower than the corresponding packet drop thresholds.

The update of the congestion window depends on whether a loss or ECN or neither occurs. The pseudo code below describes actions taken in case of the different events.

```
    on congestion event(qdelay):
      # Either loss or ECN mark is detected
      in_fast_increase = false
      cwnd_last_max = cwnd
      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)
      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()
```

The methods are further described in detail below.

4.1.2.1. Congestion window update

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 was described earlier.

Pseudo code for the update of the congestion window is found below.

```
update_cwnd(bytes_newly_acked):
  # additional scaling factor to slow down closer to target
  # The min scale factor is 0.2 to avoid that the congestion window
  #  growth is stalled when cwnd is close to cwnd_last_max
  scale_t = max(0.2,min(1.0,(4*(cwnd-cwnd_last_max)/cwnd_i)^2))

  # in fast increase ?
  if (in_fast_increase)
    if (qdelay_trend >= 0.2)
      # incipient congestion detected, exit fast increase
      in_fast_increase = false
      cwnd_last_max = cwnd
    else
      # no congestion yet, increase cwnd
      cwnd = cwnd+bytes_newly_acked*scale_t
      return

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

  gain_t = GAIN
  # adapt only increase based on scale
  if (off_target_t > 0)
    gain_t *= max(0.0, (1 - qdelay_trend/ 0.2)) * scale_t

  # increase/decrease the congestion window
  # off_target can be positive or negative
  cwnd += gain_t * off_target_t * bytes_newly_acked * MSS / cwnd
  # 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)
```

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

In fast increase state the congestion window is increased with the number of newly acknowledged bytes scaled by a scale factor that depends on the relation between CWND and the last known maximum value of CWND (cwnd_last_max).

The congestion window growth when in_fast_increase is false is dictated by the relation between qdelay and qdelay_target, also here a scale factor is applied to limit the congestion window growth when cwnd gets close to cwnd_last_max. The scale factor makes the

congestion window grow in a similar way as is the case with the Cubic congestion control algorithm i.e. a slow increase around the last known maximum value.

SCReAM calculates the GAIN in a similar way to what is specified in [RFC6817]. There are however a few differences.

- [RFC6817] specifies a constant GAIN, this specification however limits the gain when CWND is increased dependent on near congestion state and the relation to the last known max CWND value.

- [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 over-estimation of the throughput after for example, idle periods. An additional MAX_BYTES_IN_FLIGHT_HEAD_ROOM allows for a slack, to allow for a certain amount of media coder output rate variability.

4.1.2.2. Competing flows compensation

It is likely that a flow using SCReAM algorithm will have to share congested bottlenecks with other flows that use a more aggressive congestion control algorithm. SCReAM takes care of such situations by adjusting the qdelay_target.

```
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(100))
  # Compensation for competing traffic
  if (qdelay_norm_var_t < 0.16)
    # Compute average
    qdelay_norm_avg_t = AVERAGE(qdelay_norm_history(20))
    # Update target qdelay
    qdelay_target = qdelay_norm_avg_t*QDELAY_TARGET_LO*1.1
    qdelay_target = min(QDELAY_TARGET_HI, qdelay_target)
    qdelay_target = max(QDELAY_TARGET_LO, qdelay_target)
```

The qdelay_target is adjusted according to the qdelay_norm_avg_t whenever qdelay_norm_var_t is below a given value. The condition to update qdelay_target is fulfilled if qdelay_norm_var_t < 0.16.

A low qdelay_norm_avg_t value indicates that the qdelay does not change rapidly. It is desired avoid the case that the qdelay target is increased due to self-congestion, indicated by a changing qdelay and consequently an increased qdelay_norm_var_t. Still it should be possible to increase the qdelay target if the qdelay continues to be high. This is a simple function with a certain risk of both false positives and negatives but it manages competing FTP flows reasonably well at the same time as it has proven to avoid accidental increased qdelay target in simulated LTE test cases.

4.1.2.3. Lost packets detection

Lost packets dectection is based on the received sequence number list. A reordering window should be applied to avoid that packet reordering triggers loss events.
The reordering window is specified as a time unit, similar to the ideas behind RACK (Recent ACKnowledgement) [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 a 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 higher sequence number was ackelowledged.

4.1.2.4. 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:

- Bitrate variations: The media frame size is always varying to a larger or smaller extent. A strict rule as the one given above will have the effect that the media bitrate will have difficulties to increase as the congestion window puts a too hard restriction on the media frame size variation. This can lead to occasional queuing of RTP packets in the RTP packet queue that will further prevent bitrate increase.

- Reverse (feedback) path congestion: Especially in transport over buffer-bloated networks, the one way delay in the reverse direction may jump due to congestion. The effect of this is that the acknowledgements are delayed with the result that the self-

clocking is temporarily halted, even though the forward path is not congested.

The send window is adjusted depending on qdelay and its relation to the qdelay target and the relation between the congetsion 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 fast when no congestion is detected while still being able to give a 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 pseudo code below.

```
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
```

The send window is updated whenever an RTP packet is transmitted or an RTCP feedback messaged is received. More details around sender transmission control and packet pacing is found in Appendix A.3.

4.1.2.5. Resuming fast increase

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

4.1.3. Media rate control

The media rate control algorithm is executed at regular intervals 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, this 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 and a sufficient amount of data to send in order to keep the data path busy. A too cautious setting leads to possible under-utilization of network capacity and that the flow is starved out by other, more

opportunistic traffic, on the other hand a too aggressive setting leads to extra jitter.

A variable 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). The target_bitrate_min should be chosen to a low enough value to avoid that RTP packets are queued up when the network throughput becomes low. The sender should be equipped with a mechanism that discards RTP packets in cases the network throughput becomes very low and RTP packets are excessively delayed.

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

- rate_transmit: The measured transmit bitrate.

- rate_ack: The ACKed bitrate, i.e. the volume of ACKed bits per time unit.

Both estimates are updated every 200ms.

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 may itself 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 event has occurred and if the congestion control is in fast increase or not.

```
   # The target_bitrate is updated at a regular interval according
   # to RATE_ADJUST_INTERVAL

   on loss:
      target_bitrate_last_max = target_bitrate
      target_bitrate = max(BETA_R* target_bitrate, TARGET_BITRATE_MIN)
      exit

   if (in_fast_increase = true)
      scale_t = (target_bitrate - target_bitrate_last_max)/
        target_bitrate_last_max
      increment_t = RAMP_UP_SPEED*RATE_ADJUST_INTERVAL*
                 (1.0-min(1.0, qdelay_trend/0.2))
      # Value 0.2 as the bitrate should be allowed to increase
      #  at least slowly --> avoid locking the rate to
      #  target_bitrate_last_max
      increment_t *= max(0.2, min(1.0, (scale_t*4)^2))
      target_bitrate += increment_t
      target_bitrate *= (1.0- PRE_CONGESTION_GUARD*qdelay_trend)
   else
      current_rate_t = max(rate_transmit, rate_ack)
      pre_congestion = min(1.0, max(0.0, qdelay_fraction_avg-0.3)/0.7)
      pre_congestion += qdelay_trend
      target_bitrate=current_rate_t*(1.0-PRE_CONGESTION_GUARD*
           pre_congestion)-TX_QUEUE_SIZE_FACTOR *rtp_queue_size
   end

   rate_media_limit = max(br, max(rate_media,rtp_rate_median))
   rate_media_limit *= (2.0-1.0*qdelay_trend_mem)
   target_bitrate = min(target_bitrate, rate_media_limit)
   target_bitrate = min(TARGET_BITRATE_MAX,
      max(TARGET_BITRATE_MIN,target_bitrate))
```

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, a rate reduction pro actively avoids that RTP packets are queued up when the transmit rate decreases due to the reduced congestion window. An ECN event does not cause any action, the reason to this is that the congestion window is reduced less due to ECN events than loss events, the effect is thus that the expected additional RTP queuing delay due to ECN events is so small that an additional decrease in media rate is not warranted.

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. A too short interval has shown to make the video coder internal rate control loop more unstable, a too long interval makes the overall congestion control sluggish.

When in fast increase state (in_fast_increase=true), the bitrate increase is given by the desired ramp-up speed (RAMP_UP_SPEED) and is limited by the relation between the current bitrate and the last known max bitrate. Furthermore an increased qdelay trend limits the bitrate increase, an allowed increment is computed based on the congestion level (given by qdelay_trend) and the relation to target_bitrate_last_max. The target_bitrate is reduced if congestion is detected. The setting of RAMP_UP_SPEED depends on preferences, a high setting such as 1000kbps/s makes it possible to quickly get high quality media, this is however at the expense of a higher risk of jitter, which can manifest itself as e.g. choppy video rendering.

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 to the current value of target_bitrate if in_fast_increase was true the last time the bitrate was updated. Additionally, a pre-congestion indicator is computed and the rate is adjusted accordingly.

In cases where input stimuli to the media encoder is static, for instance in "talking head" scenarios, the target bitrate is not always fully utilized. This may cause undesirable oscillations in the target bitrate in the cases where the link throughput is limited and the media coder input stimuli changes between static and varying. To overcome this issue, the target bitrate is capped to be less than a given multiplier of a median value of the history of media coder output bitrates, rate_media_limit. A multiplier is applied to rate_media_limit, depending on congestion history. The target_bitrate is then limited by this rate_media_limit.

Finally the target_bitrate is enforced to be 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 and qdelay_fraction_avg. As these parameters are used also in the network congestion control one may suspect that 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 and qdelay_fraction_avg 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 faster rate increase at the expense of more jitter.

4.1.3.1. FEC and packet overhead considerations

The target bitrate given by SCReAM depicts the bitrate including RTP and FEC overhead. Therefore it is necessary that the media encoder takes this overhead into account when the media bitrate is set. It is not strictly necessary to make a 100% perfect compensation for the overhead as the SCReAM algorithm will inherently compensate moderate errors. Under-compensation for the overhead has the effect that the jitter will increase somewhat while overcompensation will have the effect that the bottleneck link becomes under-utilized.

4.2. SCReAM Receiver

The simple task of the SCReAM receiver is to feedback acknowledgements of received packets and total ECN count to the SCReAM sender, in addition, the reveive time of the RTP packet with the highest sequence number is echoed back. Upon reception of each RTP packet the receiver will simply maintain enough information to send the aforementioned values to the SCReAM sender via RTCP transport layer feedback message. The frequency of the feedback message depends on the available RTCP bandwidth. More details of the feedback and the frequency is found in Appendix A.4.

5. Discussion

This section covers a few discussion points

- Clock drift: SCReAM can suffer from the same issues with clock drift as is the case with LEDBAT [RFC6817]. Section A.2 in said RFC however describes ways to mitigate issues with clock drift.

6. Implementation status

[Editor's note: Please remove the whole section before publication, as well reference to RFC 6982]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was

supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see it".

6.1. OpenWebRTC

The SCReAM algorithm has been implemented in the OpenWebRTC project [OpenWebRTC], an open source WebRTC implementation from Ericsson Research. This SCReAM implementation is usable with any WebRTC endpoint using OpenWebRTC.

- Organization : Ericsson Research, Ericsson.
- Name : OpenWebRTC gst plug-in.
- Implementation link : The GStreamer plug-in code for SCReAM can be found at github repository [SCReAM-Implementation] The wiki (https://github.com/EricssonResearch/openwebrtc/wiki) contains required information for building and using OpenWebRTC.
- Coverage : The code implements [I-D.ietf-rmcat-scream-cc]. The current implementation has been tuned and tested to adapt a video stream and does not adapt the audio streams.
- Implementation experience : The implementation of the algorithm in the OpenWebRTC has given great insight into the algorithm itself and its interaction with other involved modules such as encoder, RTP queue etc. In fact it proves the usability of a self-clocked rate adaptation algorithm in the real WebRTC system. The implementation experience has led to various algorithm improvements both in terms of stability and design. The current implementation use an n_loss counter for lost packets indication, this is subject to change in later versions to a list of received RTP packets.
- Contact : irc://chat.freenode.net/openwebrtc

6.2. A C++ Implementation of SCReAM

- Organization : Ericsson Research, Ericsson.
- Name : SCReAM.
- Implementation link : A C++ implementation of SCReAM is also available [SCReAM-Cplusplus_Implementation] The code includes full support for congestion control, rate control and multi stream handling, it can be integrated in web clients given the addition of extra code to implement the RTCP feedback and RTP queue(s). The code also includes a rudimentary implementation of a simulator. The current implementation use an n_loss counter for lost packets indication, this is subject to change in later versions to a list of received RTP packets.
- Coverage : The code implements [I-D.ietf-rmcat-scream-cc]
- Contact : ingemar.s.johansson@ericsson.com

7. Acknowledgements

We would like to thank the following persons for their comments, questions and support during the work that led to this memo: Markus Andersson, Bo Burman, Tomas Frankkila, Frederic Gabin, Laurits Hamm, Hans Hannu, Nikolas Hermanns, Stefan Haakansson, Erlendur Karlsson, Daniel Lindstroem, Mats Nordberg, Jonathan Samuelsson, Rickard Sjoeberg, Robert Swain, Magnus Westerlund, Stefan Aalund. Many additional thanks to chairs Karen and Mirja for patiently reading, suggesting improvements and also for asking all the difficult but necessary questions. Thanks to Stefan Holmer and Xiaoqing Zhu for the review.

8. IANA Considerations

A new RFC4585 transport layer feedback message needs to be standardized.

9. 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.

10. Change history

A list of changes:

- WG-02 to WG-03: Review comments from Stefan Holmer and Xiaoqing Zhu addressed, owd changed to qdelay for clarity. Added appendix section with RTCP feedback requirements, including a suggested basic feedback format based Loss RLE report block and the Packet Receipt Times blocks in [RFC3611]. Loss detection added as a section. Transmission scheduling and packet pacing explained in appendix. Source quench semantics added to appendix.

- WG-01 to WG-02: Complete restructuring of the document. Moved feedback message to a separate draft.

- WG-00 to WG-01 : Changed the Source code section to Implementation status section.

- -05 to WG-00 : First version of WG doc, moved additional features section to Appendix. Added description of prioritization in SCReAM. Added description of additional cap on target bitrate

- -04 to -05 : ACK vector is replaced by a loss counter, PT is removed from feedback, references to source code added

- -03 to -04 : Extensive changes due to review comments, code somewhat modified, frame skipping made optional

- -02 to -03 : Added algorithm description with equations, removed pseudo code and simulation results

- -01 to -02 : Updated GCC simulation results

- -00 to -01 : Fixed a few bugs in example code

11. References

11.1. 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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6817>.

11.2. Informative References

[I-D.ietf-rmcat-app-interaction]
Zanaty, M., Singh, V., Nandakumar, S., and Z. Sarker, "RTP Application Interaction with Congestion Control", draft-ietf-rmcat-app-interaction-01 (work in progress), October 2014.

[I-D.ietf-rmcat-cc-codec-interactions]
Zanaty, M., Singh, V., Nandakumar, S., and Z. Sarker, "Congestion Control and Codec interactions in RTP Applications", draft-ietf-rmcat-cc-codec-interactions-01 (work in progress), October 2015.

[I-D.ietf-rmcat-coupled-cc]
Islam, S., Welzl, M., and S. Gjessing, "Coupled congestion control for RTP media", draft-ietf-rmcat-coupled-cc-00 (work in progress), September 2015.

[I-D.ietf-rmcat-scream-cc]
Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", draft-ietf-rmcat-scream-cc-02 (work in progress), October 2015.

[I-D.ietf-rmcat-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", draft-ietf-rmcat-wireless-tests-01 (work in progress), November 2015.

[Khademi_alternative_backoff_ECN]
"TCP Alternative Backoff with ECN (ABE)", <https://tools.ietf.org/html/draft-khademi-alternativebackoff-ecn-00>.

[OpenWebRTC]
"Open WebRTC project.", <http://www.openwebrtc.io/>.

[PACKET_CONSERVATION]
"Congestion Avoidance and Control", 1988.

[QoS-3GPP]
TS 23.203, 3GPP., "Policy and charging control architecture", June 2011, <http://www.3gpp.org/ftp/specs/archive/23_series/23.203/23203-990.zip>.

[RACK] "RACK: a time-based fast loss detection algorithm for TCP", <https://http://tools.ietf.org/id/draft-cheng-tcpm-rack-00.txt>.

[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, <http://www.rfc-editor.org/info/rfc3611>.

[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, <http://www.rfc-editor.org/info/rfc6679>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <http://www.rfc-editor.org/info/rfc6982>.

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

[SCReAM-Cplusplus_Implementation]
"C++ Implementation of SCReAM",
<https://github.com/EricssonResearch/scream>.

[SCReAM-Implementation]
"SCReAM Implementation",
<https://github.com/EricssonResearch/openwebrtc-gst-plugins>.

[TFWC] University College London, "Fairer TCP-Friendly Congestion Control Protocol for Multimedia Streaming", December 2007, <http://www-dept.cs.ucl.ac.uk/staff/M.Handley/papers/tfwc-conext.pdf>.

Appendix A. Additional information

A.1. Stream prioritization

The SCReAM algorithm makes a good distinction between network congestion control and the media rate control, an RTP queue queues up RTP packets pending transmission. This is easily extended to many streams, in which case 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 applied in [I-D.ietf-rmcat-coupled-cc] can be used. The implementation of SCReAM use something that is referred to as credit based scheduling. Credit based scheduling is for instance implemented in IEEE 802.17. The short description is that credit is accumulated by queues as they wait for service and are spent while the queues are being services.

For instance, if one queue is allowed to transmit 1000bytes, then a credit of 1000bytes is allocated to the other unscheduled queues. This principle can be extended to weighted scheduling in which case the credit allocated to unscheduled queues depends on the weight allocation.

A.2. Computation of autocorrelation function

The autocorrelation function is computed over a vector of values.

Let x be a vector constituting N values, the biased autocorrelation function for a given lag=k for the vector x is given by .

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

A.3. Sender transmission control and packet pacing

RTP packet transmission is allowed whenever the size of the next RTP packet in the sender queue is less than or equal to send window. As explained in Section 4.1.2.4 the send window is updated whenever an RTP packet is transmitted or RTCP feedback is received, the packet transmission rate is however restricted by means of packet pacing.

Packet pacing is used in order to mitigate coalescing i.e. that packets are transmitted in bursts, with the increased risk of more jitter and potentially increased packet loss.

Packet pacing is enforced when qdelay_fraction_avg is greater than 0.1. The time interval between consecutive packet transmissions is then enforced to equal or higher than t_pace where t_pace is given by the equations below.

pace_bitrate = max (50000, cwnd* 8 / s_rtt)

t_pace = rtp_size * 8 / pace_bitrate

rtp_size is the size of the last transmitted RTP packet

A.4. RTCP feedback considerations

This section describes the requrements on the RTCP feedback to make SCReAM function well. Parts of this section may be moved to a separate draft. First is described the requrements on the feedback elements, second is decribed the requirements on the feedback intensity to keep SCReAM self-clocking and rate control loops function properly.

A.4.1. Requirements on feedback elements

SCReAM requires the following elements for its basic functionality, i.e only including features that are sctrictly necessary in order to make SCReAM function. ECN is not included as basic functionality as it regarded as an additional feature that is not strickly necessary even though it can improve quality of experience quite considerably.

- A list of received RTP packets. This list should be suffciently long to cover all received RTP packets. This list may be realized with the Loss RLE report block in [RFC3611].

- A wall clock timestamp corresponding to the received RTP packet with the highest sequence number is required in order to compute the queueing delay. This can be realized by means of the Packet Receipt Times Report Block in [RFC3611]. begin_seq should be set

to the highest received (possibly wrapped around) sequence number, end_seq should be set to begin_seq+1 % 65536. The timestamp clock may be set according to the specification i.e equal to the RTP timestamp clock. Detailed individual packet receive times is not necessary as SCReAM does currently not describe how this can be used.

The basic feedback needed for SCReAM involves the use of the Loss RLE report block and the Packet Receipt Times block defined in Figure 2.

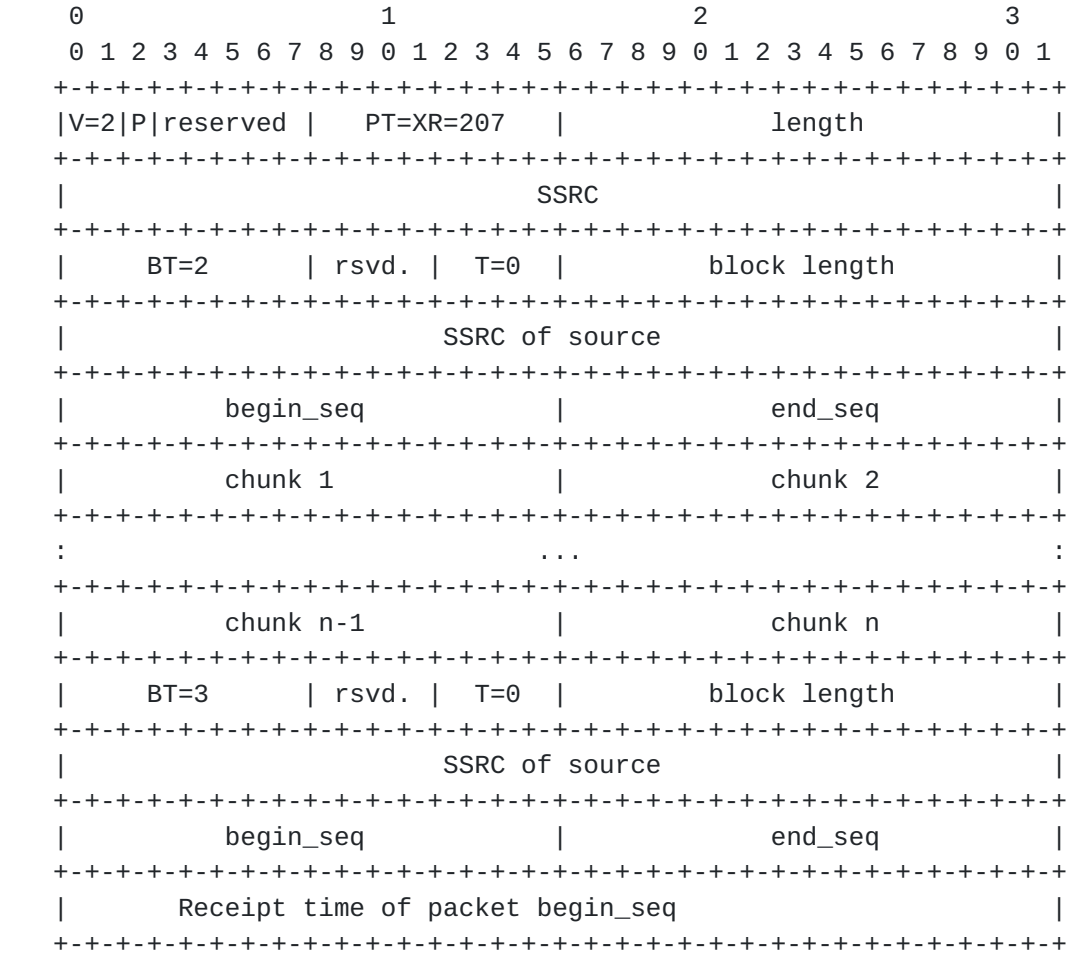

Figure 2: Basic feedback message for SCReAM

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

Additional feedback elements that can improve the performance of SCReAM are:

- Accumulated number of ECN-CE marked packets (n_ECN). This can for instance be realized with the ECN Feedback Report Format in [RFC6679]. The given feedback report format is actually a slight overkill as SCReAM would do quite well with only an 8 bit counter that increments by one for each received packet with the ECE-CE code point set. The more bulky format may be nevertheless be useful for e.g ECN black-hole detection.

- Source quench bit (Q): Makes it possible to request the sender to reduce its congestion window. This is useful if WebRTC media is received from many hosts and it becomes necessary to balance the bitrates between the streams. This can currently not be realized with any standardized feedback format.

A.4.2. Requirements on feedback intensity

SCReAM benefits from a relatively frequent feedback. Experiments have shown that a feedback rate roughly equal to the frame rate gives a stable self-clocking and robustness against loss of feedback. With a maximum bitrate of 1500kbps the RTCP feedback overhead is in the range 10-15kbps with reduced size RTCP [RFC5506], including IP and UDP framing and a reasonable compact RTCP feedback format. In other words the RTCP overhead is quite modest and should not pose a problem in the general case. Other solutions may be required in highly asymmetrical link capacity cases. Worth notice is that SCReAM can work with as low feedback rates as once every 200ms, this however comes with a higher sensitivity to loss of feedback and also a potential reduction in throughput.

SCReAM works with AVPF regular mode, immediate or early mode is not required by SCReAM but may nontheless be useful for e.g CCM messages 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].

The feedback interval is somewhat depending on the media bitrate. At low bitrates it is sufficient with a feedback interval of 100 to 200ms, while at high bitrates a feedback interval of ~20ms is to prefer.

This leads to a feedback rate according to the following equation

rate_fb = min(50,max(10,rate_media/20000))

rate_media is the RTP media bitrate expressed in [bits/s], rate_fb is the feedback rate expressed in [packets/s]. Converted to feedback interval we get

fb_int = 1.0/min(50,max(10,rate_media/20000))

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

A.5. Q-bit semantics (source quench)

The Q bit in the feedback is set by a receiver to signal that the sender should reduce the bitrate. The sender will in response to this reduce the congestion window with the consequence that the video bitrate decreases. A typical use case for source quench is when a receiver receives streams from sources located at different hosts and they all share a common bottleneck, typically it is difficult to apply any rate distribution signaling between the sending hosts. The solution is then that the receiver sets the Q bit in the feedback to the sender that should reduce its rate, if the streams share a common bottleneck then the released bandwidth due to the reduction of the congestion window for the flow that had the Q bit set in the feedback will be grabbed by the other flows that did not have the Q bit set. This is ensured by the opportunistic behavior of SCReAM's congestion control. The source quench will have no or little effect if the flows do not share the same bottleneck.

The reduction in congestion window is proportional to the amount of SCReAM RTCP feedback with the Q bit set, the below steps outline how the sender should react to RTCP feedback with the Q bit set. The reduction is done once per RTT. Let :

- n = Number of received RTCP feedback messages in one RTT
- n_q = Number of received RTCP feedback messages in one RTT, with Q bit set.

The new congestion window is then expressed as:

cwnd = max(MIN_CWND, cwnd*(1.0-0.5* n_q /n))

Note that CWND is adjusted at most once per RTT. Furthermore The CWND increase should be inhibited for one RTT if CWND has been decreased as a result of Q bits set in the feedback.

The required intensity of the Q-bit set in the feedback in order to achieve a given rate distribution depends on many factors such as RTT, video source material etc. The receiver thus need to monitor the change in the received video bitrate on the different streams and adjust the intensity of the Q-bit accordingly.

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