RMCAT WG
Internet-Draft
Intended status: Experimental
Expires: January 7, 2016

I. Johansson
Z. Sarker
Ericsson AB

July 6, 2015

Self-Clocked Rate Adaptation for Multimedia
draft-ietf-rmcat-scream-cc-01

Abstract

This memo describes a rate adaptation algorithm for conversational video services. 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 January 7, 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]. Because neither TCP nor LEDBAT was designed for interactive realtime media, a few extra features are needed to make the concept work well within this context. This memo describes these extra features.

1.1. Wireless (LTE) access properties

[I-D.ietf-rmcat-wireless-tests] introduces 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 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. In addition, the grace time, i.e. allowed reaction time from the time that the congestion is detected until a reaction in terms of a rate reduction is effected, is generally very short, in the order of one RTT (Round Trip Time).

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 concepts like self-clocking used in TFWC [TFWC] and follows packet conservation principles. The packet conservation principle is described as an important key-factor behind the protection of networks from congestion [FACK].

The packet conservation principle is realized by including an indication of the highest received sequence number in the feedback, see Section 5, from the receiver back to the sender, the sender keeps a list of transmitted packets and their respective sizes. This information is then used to determine how many bytes can be transmitted. 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]. This ensures that the e2e 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. These will be briefly described in sections Section 3.1 to Section 3.3.

The rate adaptation solution constitutes three parts- congestion control, transmission scheduling and media rate adaptation. All these three parts reside at the sender side. The receiver side algorithm is very simple in comparison as it only generates acknowledgements to received RTP packets.

3.1. Congestion Control

The 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 transmission scheduling.

The SCReAM congestion control method, uses LEDBAT [RFC6817] to measure the OWD (one way delay). The SCReAM sender calculates the congestion window based on the feedback from SCReAM receiver. The congestion window is allowed to increase if the OWD is below a predefined target, otherwise the congestion window decreases. The delay target is typically set to 50-100ms. This ensures that the OWD is kept low on the average. The reaction to loss events is similar to that of loss based TCP, i.e. an instant reduction of CWND.

LEDBAT is designed with file transfers as main use case which means that the algorithm must be modified somewhat to work with rate-limited sources such as video. The modifications are

- Congestion window validation techniques. These are similar in action as the method described in [I-D.ietf-tcpm-newcwv].

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

- Adaptive delay target. This helps the congestion control to compete with FTP traffic to some degree.

3.2. Transmission Scheduling

Transmission scheduling limits the output of data, given by the relation between the number of bytes in flight and the congestion window similar to TCP. Packet pacing is used to mitigate issues with coalescing that may cause increased jitter and/or packet loss in the media traffic.

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 queues. 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 video frames or frame skipping in order to ensure that the RTP queues are drained quickly. Frame skipping means that the frame rate is temporarily reduced. Discarding of old video frames is a more efficient way to reduce media latency than frame skipping but it comes with a requirement to repair codec state, frame skipping is thus to prefer as a first remedy. Frame skipping is described as an optional to implement feature in this specification.

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 and the video rate adaptation.

Figure 1 shows the functional overview of a SCReAM sender. The RTP application interaction with congestion control is described in [I-D.ietf-rmcat-app-interaction]. Here we use a more decomposed version of the implementation model in the sense that the RTP packets may be queued up in the sender, the transmission of these RTP packets is controlled by a transmission scheduler. A SCReAM sender implements rate control and a queue for each media type or source, where RTP packets containing encoded media frames are temporarily stored for transmission, the figure shows the details for when two video sources (a.k.a streams) are used.

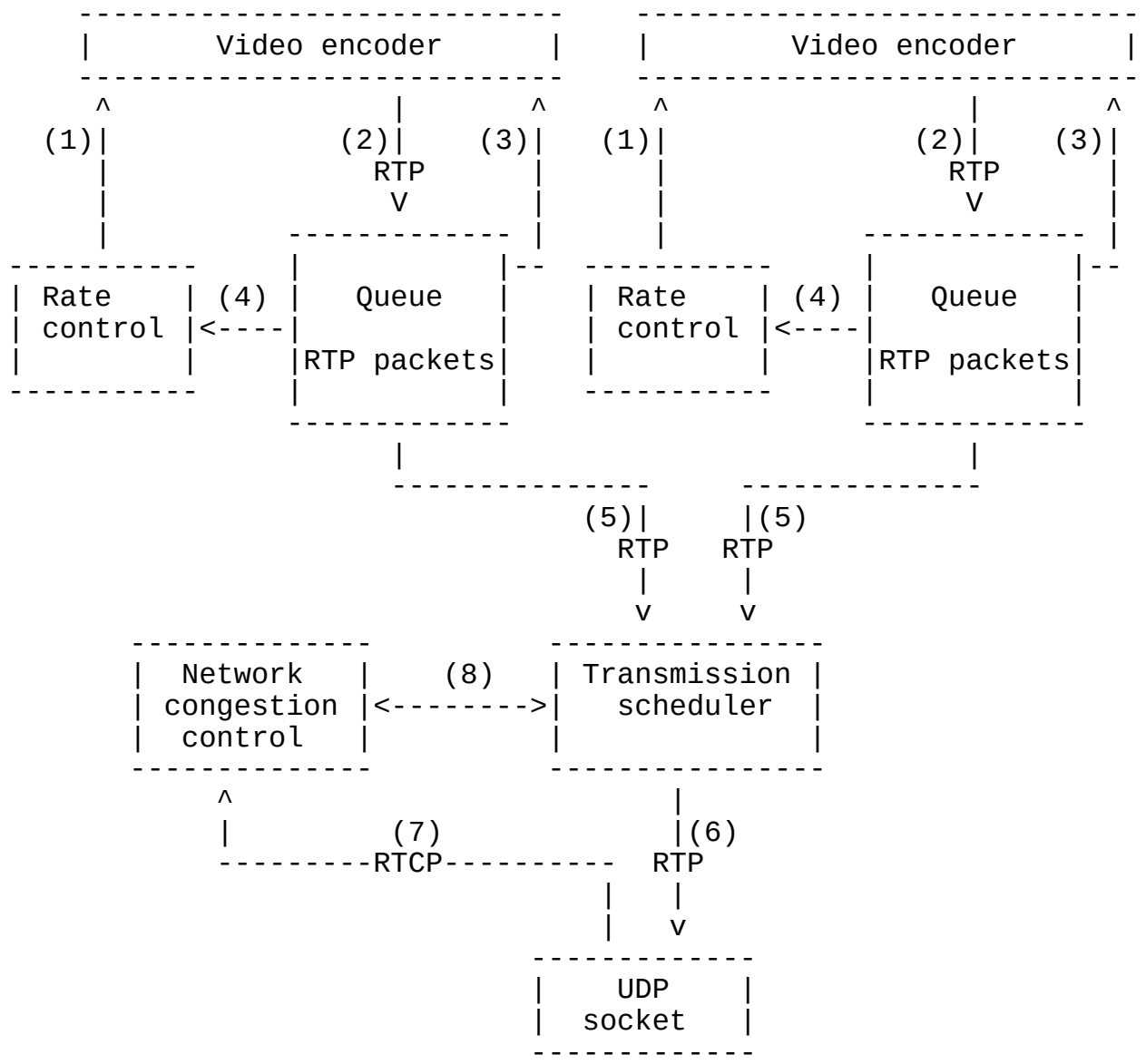

Figure 1: SCReAM sender functional view

Video frames are encoded and forwarded to the queue (2). The media rate adaptation adapts to the size of the RTP queue and controls the video bitrate (1). The RTP packets are picked from each queue based on some defined priority order or simply in a round robin fashion (5). A transmission scheduler takes care of the transmission of RTP packets, to be written to the UDP socket (6). In the general case all media must go through the transmission scheduler and is allowed to be transmitted if the number of bytes in flight is less than the congestion window. Audio frames can however be allowed to be transmitted immediately as audio is typically low bitrate and thus contributes little to congestion, this is something that is left as an implementation choice. RTCP packets are received (7) and the information about bytes in flight and congestion window is exchanged between the network congestion control and the transmission scheduler (8).

4.1.1. Constants and Parameter values

A set of constants are defined in Table 1, state variables are defined in Table 2. And finally, local variables are described in Table 3.

An init value [] indicates an empty array.

Constant	Explanation	Value
OWD_TARGET_LO	Min OWD target	0.1s
OWD_TARGET_HI	Max OWD target	0.4s
MAX_BYTES_IN_FLIGHT_HEAD_ROOM	Headroom for limitation of CWND	1.1
GAIN	Gain factor for congestion window adjustment	1.0
BETA	CWND scale factor due to loss event	0.6
BETA_R	Target rate scale factor due to loss event	0.8
BYTES_IN_FLIGHT_SLACK	Additional slack [%] to the congestion window	10%
RATE_ADJUST_INTERVAL	Interval between video bitrate adjustments	0.1s
FRAME_PERIOD	Video coder frame period [s]	
TARGET_BITRATE_MIN	Min target_bitrate [bps]	
TARGET_BITRATE_MAX	Max target_bitrate [bps]	
RAMP_UP_TIME	Timespan [s] from lowest to highest bitrate	10s
PRE_CONGESTION_GUARD	Guard factor against early congestion onset. A higher value gives less jitter possibly at the expense of a lower video bitrate.	0.0..0.2
TX_QUEUE_SIZE_FACTOR	Guard factor against RTP queue buildup	0.0..2.0

Table 1: Constants

Variable	Explanation	Init value
owd_target	OWD target	OWD_TARGET_LO
owd_fraction_avg	EWMA filtered owd_fraction	0.0
owd_fraction_hist	Vector of the last 20 owd_fraction	[]
owd_trend	OWD trend, indicates incipient congestion	0.0
owd_trend_mem	Low pass filtered version of owd_trend	0.0
owd_norm_hist	Vector of the last 100 owd_norm	[]
mss	Maximum segment size = Max RTP packet size [byte]	1000
min_cwnd	Minimum congestion window [byte]	2*MSS
in_fast_start	True if in fast start state	true
cwnd	Congestion window [byte]	min_cwnd
cwnd_i	Congestion window inflection point	1
bytes_newly_acked	The number of bytes that was acknowledged with the last received acknowledgement i.e. bytes acknowledged since the last CWND update [byte]. Reset after a CWND update	0
send_wnd	Upper limit of how many bytes that can be transmitted [byte]. Updated when CWND is updated and when RTP packet is transmitted	0
t_pace	Approximate	0.001

	estimate of inter-packet transmission interval [s], updated when RTP packet transmitted		
age_vec	A vector of the last 20 RTP packet queue delay samples	[]	
frame_skip_intensity	Indicates the intensity of the frame skips	0.0	
since_last_frame_skip	Number of video frames since the last skip	0	
consecutive_frame_skips	Number of consecutive frame skips	0	
target_bitrate	Video target bitrate [bps]	TARGET_BITRATE_MIN	
target_bitrate_i	Video target bitrate inflection point i.e. the last known highest target_bitrate during fast start. Used to limit bitrate increase close to the last know congestion point	1	
rate_transmit	Measured transmit bitrate [bps]	0.0	
rate_acked	Measured throughput based on received acknowledgements [bps]	0.0	
rate_rtp	Measured bitrate from the media encoder [bps]	0.0	
rate_rtp_median	Median value of rate_rtp, computed over more than 10s [bps]	0.0	
s_rtt	Smoothed RTT [s], computed similar	0.0	

	to method depicted in [RFC6298]		
rtp_queue_size	Size of RTP packets in queue [bits]	0	
rtp_size	Size of the last transmitted RTP packets [byte]	0	
frame_skip	Skip encoding of video frame if true	false	

Table 2: State variables

Variable	Explanation
owd	OWD = One way delay with base delay subtracted [s]. This is an estimate of the network queueing delay.
owd_fraction	OWD as a fraction of the OWD target
owd_norm	OWD normalized to OWD_TARGET_LO
owd_norm_mean	Average OWD norm over the last 100 samples
owd_norm_mean_sh	Average OWD norm over the last 20 samples
owd_norm_var	OWD norm variance over the last 100 samples
off_target	Relation between OWD and OWD target
scl_i	A general scalefactor that is applied to the CWND or target_bitrate increase
x_cwnd	Additional increase of CWND, used when send_wnd is computed
pace_bitrate	The allowed RTP packet transmission rate, used in the computation of t_pace [bps]
age_avg	Average RTP queue delay [s]
increment	Allowed target_bitrate increase
current_rate	Max of rate_transmit and rate_acked

Table 3: Local temporary variables

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.

- o Transmission scheduling 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 it keeps a list of transmitted RTP packets and their respective sending times (wall-clock time). The feedback indicates the highest received RTP sequence number and a timestamp (wall-clock time) when it was received. In addition, an ACK list is included to make it possible to determine lost packets.

4.1.2.1. Congestion window update

The congestion window is computed from the one way (extra) delay estimates (OWD) that are obtained from the send and received timestamp of the RTP packets. LEDBAT [RFC6817] explains the details of the computation of the OWD. An OWD sample is obtained for each received acknowledgement. No smoothing of the OWD samples occur, however some smoothing occurs anyway as the computation of the CWND is in itself a low pass filter function.

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. As an example: If RTP packet was sequence number SN with transmitted and the last ACK indicated 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.

CWND is updated differently depending on whether the congestion control is in fast start or not and if a loss event is detected. A Boolean variable in_fast_start indicates if the congestion is in fast start state.

A loss event indicates one or more lost RTP packets within an RTT. This is detected by means of inspection for holes in the sequence number space in the acknowledgements with some margin for possible packet reordering in the network. As an alternative, a timer for loss detection similar to TCP RACK may be used.

Below is described the actions when an acknowledgement from the receiver is received.

bytes_newly_acked is updated.

The OWD fraction and an average of it are computed as

owd_fraction = owd/owd_target

owd_fraction_avg = 0.9* owd_fraction_avg + 0.1* owd_fraction

The OWD fraction is sampled every 50ms and the last 20 samples are stored in a vector (owd_fraction_hist). This vector is used in the computation of an OWD trend that gives a value between 0.0 and 1.0 depending on how close to congestion it is. The OWD trend is calculated as follows

Let R(owd_fraction_hist,K) be the autocorrelation function of owd_fraction_hist at lag K. The 1st order prediction coefficient is formulated as

a = R(owd_fraction_hist,1)/R(owd_fraction_hist,0)

The prediction coefficient a has positive values if OWD shows an increasing trend, thus an indication of congestion is obtained before the OWD target is reached. The prediction coefficient is further multiplied with owd_fraction_avg to reduce sensitivity to increasing OWD when OWD is very small. The OWD trend is thus computed as

owd_trend = max(0.0,min(1.0,a*owd_fraction_avg))

owd_trend_mem = max(0.99*owd_trend_mem, owd_trend)

The owd_trend is utilized in the media rate control and to determine when to exit slow start. owd_trend_mem is used to enforce a less aggressive rate increase after congestion events.

An off target value is computed as

off_target = (owd_target - owd) / owd_target

A temporal variable is scl_i is computed as

scl_i = max(0.2, min(1.0, (abs(cwnd-cwnd_i)/cwnd_i*4)^2))

scl_i is used to limit the CWND increase when close to the last known max value, before congestion was last detected.

The congestion window update depends on whether a loss event has occurred, and if the congestion control is if fast start or not.

On loss event:

If a loss event is detected then in_fast_start is set to false and CWND is updated according to

cwnd_i = cwnd

cwnd = max(min_cwnd,cwnd*BETA)

otherwise the CWND update continues

---

in_fast_start = true:

in_fast_start is set to false and cwnd_i=cwnd if owd_trend >= 0.2 and otherwise CWND is updated according to

cwnd = cwnd + bytes_newly_acked*scl_i

---

in_fast_start = false:

Values of off_target > 0.0 indicates that the congestion window can be increased. This is done according to the equations below.

gain = GAIN*(1.0 + max(0.0, 1.0 - owd_trend/ 0.2))

The equation above limits the gain when near congestion is detected

gain *= scl_i

This equation limits the gain when CWND is close to its last known max value

cwnd += gain * off_target * bytes_newly_acked * mss / cwnd

Values of off_target <= 0.0 indicates congestion, CWND is then updated according to the equation

cwnd += GAIN*off_target*bytes_newly_acked*mss/cwnd

The equations above are very similar to what is specified in [RFC6817]. There are however a few differences.

- o [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.

- o [RFC6817] specifies that the CWND increased is limited by an additional function controlled by a constant ALLOWED_INCREASE. This additional limitation is removed in this specification.

A number of final steps in the congestion window update procedure are outlined below

---

Resume fast start:

Fast start can be resumed in order to speed up the bitrate increase in case congestion abates. The condition to resume fast start (in_fast_start = true) is that owd_trend is less than 0.2 for 1.0 seconds or more.

Competing flows compensation, adjustment of owd_target:

Competing flows compensation is needed to avoid that flows congestion controlled by SCReAM are starved out by flows that are more aggressive in their nature. The owd_target is adjusted according to the owd_norm_mean_sh whenever owd_norm_var is below a given value. The condition to update owd_target is fulfilled if owd_norm_var < 0.16 (indicating that the standard deviation is less than 0.4). owd_target is then update as:

owd_target = min(OWD_TARGET_HI,max(OWD_TARGET_LO, owd_norm_mean_sh* OWD_TARGET_LO*1.1))

Final CWND adjustment step:

The congestion window is limited by the maximum number of bytes in flight over the last 1.0 seconds according to

cwnd = min(cwnd, max_bytes_in_flight*MAX_BYTES_IN_FLIGHT_HEAD_ROOM)

This avoids possible over-estimation of the throughput after for example, idle periods.

Finally cwnd is set to ensure that it is at least min_cwnd

cwnd = max(cwnd, MIN_CWND)

4.1.2.2. Transmission scheduling

The principle is to allow packet transmission of an RTP packet 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 video 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 video bitrate have difficulties to increase as the congestion window puts a too hard restriction on the video frame size variation, this further can lead to occasional queuing of RTP packets in the RTP packet queue that will prevent bitrate increase because of the increased RTP queue size.

- 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.

Packets are transmitted at a pace given by the send window, computed below

The send window is computed differently depending on OWD and its relation to the OWD target.

- If owd > owd_target:
  The send window is computed as
  send_wnd = cwnd-bytes_in_flight
  This enforces a strict rule that helps to prevent further queue buildup.

- If owd <= owd_target:
  A helper variable
  x_cwnd=1.0+BYTES_IN_FLIGHT_SLACK*max(0.0,
  min(1.0,1.0-owd_trend/0.5))/100.0
  is computed. The send window is computed as
  send_wnd = max(cwnd*x_cwnd, cwnd+mss)-bytes_in_flight

This gives a slack that reduces as congestion increases, BYTES_IN_FLIGHT_SLACK is a maximum allowed slack in percent. A large value increases the robustness to bitrate variations in the source and congested feedback channel issues. The possible drawback is increased delay or packet loss when forward path congestion occur.

4.1.3. Video rate control

The video rate control is operated based on the size of the RTP packet send queue and observed loss events. In addition, owd_trend is also considered in the rate control, this to reduce the amount of induced network 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).

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

- rate_transmit: The measured transmit bitrate
- rate_acked: 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_acked. The rationale behind the use of rate_acked 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_acked is used as well. This gives a more stable throughput estimate.

The bitrate is updated at regular intervals, given by RATE_ADJUST_INTERVAL and differently depending the fast start state

The rate change behavior depends on whether a loss event has occurred, and if the congestion control is if fast start or not.

On loss event:

First of all the target_bitrate is updated if a new loss event was indicated and the rate change procedure is exited.

target_bitrate_i = target_bitrate

target_bitrate = max(BETA_R* target_bitrate, TARGET_BITRATE_MIN)

If no loss event was indicated then the rate change procedure continues.

---

in_fast_start = true:

An allowed increment is computed based on the congestion level and the relation to target_bitrate_i

scl_i = (target_bitrate - target_bitrate_i)/ target_bitrate_i

increment = TARGET_BITRATE_MAX* RATE_ADJUST_INTERVAL/RAMP_UP_TIME* (1.0- min(1.0, owd_trend/0.1))

increment *= max(0.2, min(1.0, (scl_i*4)^2))

target_bitrate += increment

target_bitrate is reduced further if congestion is detected.

target_bitrate *= (1.0- PRE_CONGESTION_GUARD*owd_trend)

---

in_fast_start = false:

target_bitrate_i is updated to the current value of target_bitrate if in_fast_start was true the last time the bitrate was updated.

A pre-congestion indicator is computed as

pre_congestion = min(1.0, max(0.0, owd_fraction_avg-0.3)/0.7)

pre_congestion += owd_trend

The target bitrate is computed as

```
target_bitrate=current_rate*(1.0-
PRE_CONGESTION_GUARD*pre_congestion)-TX_QUEUE_SIZE_FACTOR
*rtp_queue_size
```

Final step:

As a final step, the target bitrate is limited such that it is kept within reasonable bounds.

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. A rate_rtp_limit is computed as

```
rate_rtp_limit = max(br, max(rate_rtp,rtp_rate_median))
```

A multiplier is applied to rate_rtp_limit, depending on congestion history

```
rate_rtp_limit *= (2.0-1.0*owd_trend_mem)
```

The target_bitrate is then limited by rate_rtp_limit

```
target_bitrate = min(target_bitrate, rate_rtp_limit)
```

Finally the target_bitrate is enforced to be within the defined min and max values

```
target_bitrate =
min(TARGET_BITRATE_MAX,max(TARGET_BITRATE_MIN,target_bitrate))
```

4.2. SCReAM Receiver

The SCReAM receiver is very simple in its implementation. The task is to feedback acknowledgements of received packets. For that purpose a set of state variables are needed, these are explained in Table 4.

One set of state variables are maintained per stream.

Variable	Explanation	Init value
rx_timestamp	The wall clock timestamp when the latest RTP packet was received	0
highest_rtp_sequence_number	The highest received sequence number	0
ack_vector	A 16 bit vector that indicates received RTP packets with a sequence number lower than highest_rtp_sequence_number	0
n_loss	An 8 bit counter for the number of lost RTP packets, separate counters are maintained for each SSRC	0
n_ECN	An 8 bit counter for the number of ECN-CE marked RTP packets, separate counters are maintained for each SSRC	0
pending_feedback	Indicates that an RTP packet was received and that an RTCP packet can be generated when RTCP timing rules permit	false
last_transmit_t	Last time an RTCP packet was transmitted, this is used to ensure that RTCP feedback is generated fairly for all streams.	-1.0

Table 4: State variables

Upon reception of an RTP packet, the state variables in Table 4 should be updated and the RTCP processing function should be notified. An RTCP packet is later generated based on the state variables, how often this is done depends on the RTCP bandwidth.

5. Feedback Message

The feedback is over RTCP [RFC3550] and is based on [RFC4585]. It is implemented as a transport layer feedback message (RTPFB), see proposed example in Figure 2. The feedback control information part (FCI) consists of the following elements.

- Highest received RTP sequence number: The highest received RTP sequence number for the given SSRC

- n_lost: Ackumulated number of lost RTP packets for the given SSRC

- Timestamp: A timestamp value indicating when the last packet was received which makes it possible to compute the one way (extra) delay (OWD).

- n_ECN: Ackumulated number of ECN-CE marked RTP packets for the given SSRC

- 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.

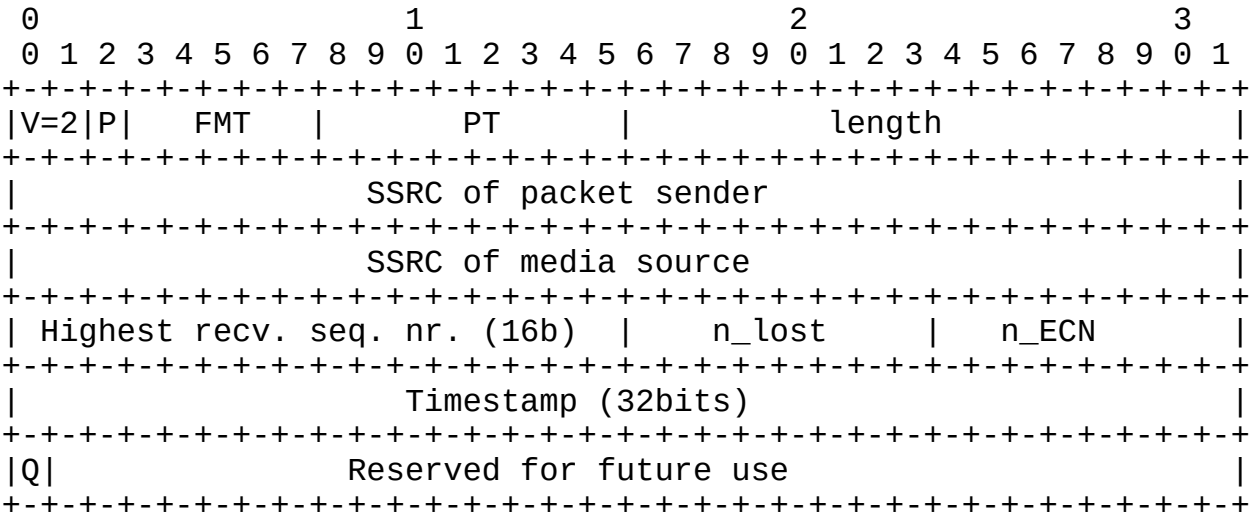

Figure 2: Transport layer feedback message

To make the feedback as frequent as possible, the feedback packets are transmitted as reduced size RTCP according to [RFC5506].

The timestamp clock time is recommended to be set to a fixed value such as 1000Hz, defined in this specification. The n_lost and n_ECN makes it possible to take necessary actions on the detection of lost and ECN marked packets.

Section 4 describes the main algorithm details and how the feedback is used.

6. Discussion

This section covers a few open discussion points

- RTCP feedback overhead: 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, including IP and UDP framing, 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.

- AVPF mode: The RTCP feedback is based on AVPF regular mode. The SCReAM feedback is transmitted as reduced size RTCP so save overhead, it is however required to transmit full compound RTCP at regular intervals, this interval can be controlled by trr-int depicted in [RFC4585].

- BETA, CWND scale factor due to loss: The BETA value is recommended to be higher than 0.5. The reason behind this is that congestion control for multimedia has to deal with a source that is rate limited. A file transfer has "unlimited" source bitrate in comparison. The outcome is that SCReAM must be a little more aggressive than a file transfer in order to not be out competed.

7. Conclusion

This memo describes a congestion control algorithm for RMCAT that it is particularly good at handling the quickly changing condition in wireless network such as LTE. The solution conforms to the packet conservation principle and leverages on novel congestion control algorithms and recent TCP research, together with media bitrate determined by sender queuing delay and given delay thresholds. The solution has shown potential to meet the goals of high link utilization and prompt reaction to congestion. The solution is realized with a new RFC4585 transport layer feedback message.

8. Open issues

A list of open issues.

- Describe how clock drift compensation is done

- Describe how FEC overhead is accounted for in target_bitrate computation

- Investigate the impact of more sparse RTCP feedback, for instance once per RTT

- Describe ECN behavior

9. 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".

9.1. 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] and is waiting to be merged with the master branch of OpebWebRTC repository (https://github.com/EricssonResearch/openwebrtc/pull/413).

However, people are encouraged to have look at it and send feedback. This wiki (https://github.com/EricssonResearch/openwebrtc/wiki) contains required information for building and using OpenWebRTC. Note that to get all the SCReAM related code and build them, one has to use the cerbero fork from DanielLindstrm' s repository (https://github.com/DanielLindstrm/cerbero/tree/scream) instead of EricssonResearch fork of cerbero.

- Coverage : The code implements [I-D.ietf-rmcat-scream-cc]. The current implementation has been tuned and tested to adapt 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. For example, improved rate increase behavior and removal of the ACK vector from the feedback message.

- Contact : irc://chat.freenode.net/openwebrtc

9.2. A C++ Implementation of SCReAM

- Organization : Ericsson Research, Ericsson.

- Name : SCReAM.

- Implementation link : A C++ implementation of SCreAM is also available which is aimed for doing quick experiments[SCReAM-Cplusplus_Implementation]. This repository also includes a rudimentary implementation of a simulator. This code can be included in other simulators like NS-3.

- Coverage : The code implements [I-D.ietf-rmcat-scream-cc]

- Contact : ingemar.s.johansson@ericsson.com, zaheduzzaman.sarker@ericsson.com

10. 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.

11. IANA Considerations

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

12. 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.

13. Change history

A list of changes:

- 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

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[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, July 2006.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, December 2012.

14.2. Informative References

[FACK] "Forward Acknowledgement: Refining TCP Congestion Control", 2006.

[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-scream-cc] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", draft-ietf-rmcat-scream-cc-00 (work in progress), May 2015.

[I-D.ietf-rmcat-wireless-tests] Sarker, Z. and I. Johansson, "Evaluation Test Cases for Interactive Real-Time Media over Wireless Networks", draft-ietf-rmcat-wireless-tests-00 (work in progress), June 2015.

[I-D.ietf-tcpm-newcwv] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to support Rate-Limited Traffic", draft-ietf-tcpm-newcwv-13 (work in progress), June 2015.

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

[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>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

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

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

[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 features

This section describes additional features. They are not required for the basic functionality of SCReAM but can improve performance in certain scenarios and topologies.

A.1. Packet pacing

Packet pacing is used in order to mitigate coalescing i.e. that packets are transmitted in bursts.

Packet pacing is enforced when owd_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.2. Stream prioritization

As mentioned in Section 4, the prioritization between several streams can be managed in many different ways. The most simple way is to pick RTP packets from the RTP queues in a round-robin fashion. For more advanced scheduling, more advanced algorithms are required. Below is described the algorithm that is implemented in the SCReAM code Section 9.

Suppose that we have two video streams, where stream 1 has priority 1.0 and stream 2 has priority 0.5. Each stream starts with a credit of 0 bytes, credit is given to streams that are not given permission to transmit at a given scheduling instant, the credit is considered in later transmission instants.

The steps below outline how transmission and scheduling of the two RTP streams can evolve. For simplicily it is assumed that the stream RTP queues contain 1200 byte packets.

1. SCReAMs send window allows transmission of 1200 bytes.

    * The stream with the highest priority is picked, in this case it is stream 1. Stream 1 thus transmit 1200 bytes.

    * Stream 2 gets its credit increased by 1200*0.5/1.0 = 600 byte and thus has a credit of 600 bytes.

2. SCReAMs send window allows transmission of another 1200 bytes.

    * Stream 2 has too little credit (600 bytes) to transmit a 1200 byte packet.

    * Stream 1 is therefore picked again as it has the highest priority and thus gets to transmit yet another 1200 byte packet.

    * Stream 2 gets its credit increased by 1200*0.5/1.0 = 600 byte and thus has a credit of 1200 bytes.

3. SCReAMs send window allows transmission of another 1200bytes.

    * Stream 2 now has enough credit (1200 bytes) to transmit a 1200 byte packet.

    * Stream 2 thus transmits a 1200 byte packet and in the process gets its credit reduced by 1200 byte and is then down to a credit of 0.

* Stream 1 gets its credit increased by 1200*1.0/0.5 = 2400 byte and thus has a credit of 2400 bytes.

4. SCReAMs send window allows transmission of another 1200 bytes.

    1. Stream 1 now has the highest credit (2400bytes).

    2. Stream 1 thus transmits a 1200 byte packet and in the process gets its credit reduced by 1200 byte and is then down to a credit of 1200 bytes.

    3. Stream 2 gets its credit increased by 1200*0.5/1.0 = 600 byte and thus has a credit of 600 bytes.

5. SCReAMs send window allows transmission of another 1200 bytes.

    1. Stream 1 still has the highest credit (1200 bytes).

    2. Stream 1 thus transmits a 1200 byte packet and in the process gets its credit reduced by 1200 byte and is then down to a credit of 0.

    3. Stream 2 gets its credit increased by 1200*0.5/1.0 = 600 byte and thus has a credit of 1200bytes.

6. SCReAMs send window allows transmission of another 1200 bytes.

    1. Stream 2 now has the highest credit (1200 bytes).

    2. Stream 2 thus transmits a 1200 byte packet and in the process gets its credit reduced by 1200 byte and is then down to a credit of 0.

    3. Stream 1 gets its credit increased by 1200*1.0/0.5 = 2400 byte and thus has a credit of 2400 bytes.

The procedure above repeats it self. In the above example it is quite easy to see that stream 1 gets to transmit 2 RTP packets for every 1 RTP packets that stream 2 gets to transmit. The very detais of the algoritm is found in the C++ code (see Section 9) in the module ScreamTx and the functions getPrioritizedStream(..), addCredit(..) and subtractCredit(..).

The above functionality works relatively well and operates with at the same speed as RTP packet transmission. There are however cases where rate limited video or very large IR frames makes the prioritization less efficient. The adjustPriorities(..) function in ScreamTx solves this issue on a longer time scale by means of an

additional compensation for deviations in the measured transmit bitrate of the individual streams.

Prioritization mechanisms of sources that may be highly variant is a relatively complicated task to achieve. The above outlined algorithm manages it to some degree but it is quite likely that the algorithm needs to be refined further.

A.3. 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.

A.4. Frame skipping

Frame skipping is a feature that makes it possible to reduce the size of the RTP queue in the cases that e.g. the channel throughput drops dramatically or even goes below the lowest possible video coder rate. Frame skipping is optional to implement as it can sometimes be difficult to realize e.g. due to lack of API function to support this.

Frame skipping is controlled by a flag frame_skip which, if set to 1 dictates that the video coder should skip the next video frame. The frame skipping intensity at the current time instant is computed according to the steps below

The queuing delay is sampled every frame period and the last 20 samples are stored in a vector age_vec

An average queuing delay is computed as a weighted sum over the samples in age_vec. age_avg at the current time instant is computed as

age_avg(n) = SUM age_vec(n-k)*w(k) k = [0..20[

w(n) are weight factors arranged to give the most recent samples a higher weight.

The change in age_avg is computed as

age_d = age_avg(n) - age_avg(n-1)

The frame skipping intensity at the current time instant n is computed as

- If age_d > 0 and age_avg > 2*FRAME_PERIOD: frame_skip_intensity = min(1.0, (age_vec(n)-2*FRAME_PERIOD)/(4* FRAME_PERIOD)

- Otherwise frame skip intensity is set to zero

The skip_frame flag is set depending on three variables

- frame_skip_intensity

- since_last_frame_skip, i.e the number of consecutive frames without frame skipping

- consecutive_frame_skips, i.e the number of consecutive frame skips

The flag skip_frame is set to 1 if any of the conditions below is met, otherwise it is set to 0.

- age_vec(n) > 0.2 && consecutive_frame_skips < 5
- frame_skip_intensity < 0.5 && since_last_frame_skip >= 1.0/ frame_skip_intensity
- frame_skip_intensity >= 0.5 && consecutive_frame_skips < (frame_skip_intensity -0.5)*10

The arrangement makes sure that no more than 4 frames are skipped in sequence, the rationale is to ensure that the input to the video encoder does not change to much, something that may give poor prediction gain.

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