SCREAM C++ CODE

Ingemar Johansson, Ericsson Research
Ingemar.s.johansson@ericsson.com

INTRO

SCReAM (**S**elf-**C**locked **R**at**e** **A**daptation for **M**ultimedia) is a congestion control algorithm devised mainly for Video.

Congestion control for WebRTC media is currently being standardized in the IETF RMCAT WG, the scope of the working group is to define requirements for congestion control and also to standardize a few candidate solutions.

SCReAM is a congestion control candidate solution for WebRTC developed at Ericsson Research and optimized for good performance in wireless access.

The algorithm is submitted to the RMCAT WG [1], a Sigcomm paper [2] and [3] explains the rationale behind the design of the algorithm in more detail. A comparison against GCC (Google Congestion Control) is shown in [4].

Unlike many other congestion control algorithms that are rate based i.e. they estimate the network throughput and adjust the media bitrate accordingly, SCReAM is self-clocked which essentially means that the algorithm does not send in more data into a network than what actually exits the network.

To achieve this, SCReAM implements a feedback protocol over RTCP that acknowledges received RTP packets. The C++ code does not assume that feedback is transmitted over RTCP, on the contrary it is possible to transmit the feedback in RTP header extensions in cases where this alternative is more beneficial.

A congestion window is determined from the feedback, this congestion window determines how many RTP packets that can be in flight i.e. transmitted by not yet acknowledged, an RTP queue is maintained at the sender side to temporarily store the RTP packets pending transmission, this RTP queue is mostly empty but can temporarily become larger when the link throughput decreases.

The congestion window is frequently adjusted for minimal e2e delay while still maintaining as high link utilization as possible. The use of self-clocking in SCReAM which is also the main principle in TCP has proven to work particularly well in wireless scenarios where the link throughput may change rapidly. This enables a congestion control which is robust to channel jitter, introduced by e.g. radio resource scheduling while still being able to respond promptly to reduced link throughput.

SCReAM is optimized using a state the art LTE system simulator for optimal performance in deployments where the LTE radio conditions are limiting. In addition, SCReAM is also optimized for good performance in simple bottleneck case such as those given in home gateway deployments.

This presentation describes the C++ implementation of SCReAM. The package is currently implemented as a Visual Studio 2010 solution but should be straightforward to port to other platforms, additional Cmake files makes it possible to compile the code on most platforms

References

[1] http://tools.ietf.org/wg/rmcat/draft-ietf-rmcat-scream-cc

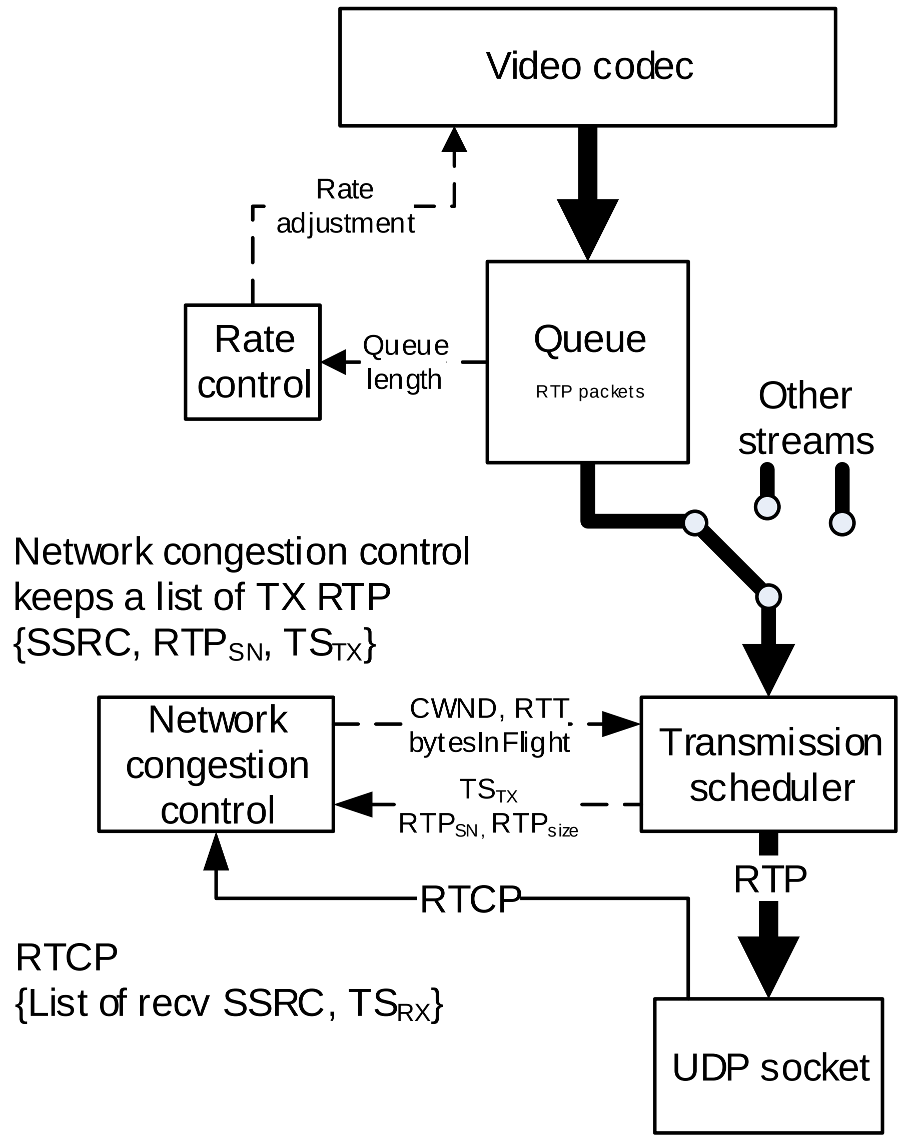
[2] Sigcomm paper http://dl.acm.org/citation.cfm?id=2631976

[3] Sigcomm presentation http://conferences.sigcomm.org/sigcomm/2014/doc/slides/150.pdf

[4] IETF RMCAT presentation, comparison against Google Congestion Control (GCC) http://www.ietf.org/proceedings/90/slides/slides-90-rmcat-3.pdf

INTRO

- SCReAM does not allow RTP packets to be transmitted directly
- Packet transmission is controlled by transmission scheduler which is guided by the network congestion control



SCReAM CLASSES

- Main SCReAM algorithm :
  - ScreamTx : SCReAM sender algorithm. Does all the intelligent congestion control magic.
  - ScreamRx : SCReAM receiver algorithm. Dumber than dumb, just records packet receive times and a list of received packets and prepares feedback elements.
- Support classes for experiment :
  - RtpQueue : Rudimentary RTP queue
  - VideoEnc : Simple model of Video encoder
  - NetQueue : Simple delay and bandwidth limitation

ScreamTx

- Implements SCReAM congestion control algorithm on sender side
- Contains a reference to class RtpQueue which needs to be replaced if ScreamTx is to be integrated in other platforms
  - RtpQueue replacement need to provide with the following functions
    - sizeOfNextRtp() , size of next RTP packet in RTP queue
    - getDelay() , queuing delay of the oldest RTP packet
    - sizeOfQueue(), aggregate size of all the RTP packets in the queue
    - sizeOfLastFrame(), aggregate size of the RTP packets in the queue with the highest RTP timestamp
- Time is counted in microseconds (uint64_t), microsecond resolution is however not required

SCREAMTX TUNING

- A few tuning parameters are listed below.
- **kEnableSbd** : Enables shared bottleneck detection and an increase of the owdTarget. This feature is useful if SCReAM congestion controlled media has to compete with more greedy traffic such as FTP. There is a certain risk that the feature can false trigger and increase the owdTarget even though there are no competing greedy traffic, with the result that the e2e delay increases. So if you suspect that your application does not need to compete with greedy traffic then set kEnableSbd=false.
- **kRampUpSpeed**: Determines how quickly the target bitrate can increase, defined in bps/s (bits per second increase per second). A high value such as 1e6 makes the video bitrate increase quick, this is however at the expense of a higher risk of unstable behavior and larger e2e delay jitter, recommended values are around 200000-500000 bps/s.
- **kLossBeta**: Dictates how much the CWND should be scaled when a loss event is detected. The SCReAM algorithm will more aggressively grab available bandwidth if this value is increased, at the expense of a higher packet loss rate.
- **kTxQueueSizeFactor**: Determines what impact an increased RTP queue size (in bytes) has on the media coder target bitrate. A low value may cause increased e2e delay for the media both due to increased RTP queue delay and increased network queuing delay. A too high value may make the algorithm yield too much to competing flows.
- **kQueueDelayGuard**: Determines the impact that an increased one way delay has on the media coder target bitrate. A low value may cause increased network queuing delay. A too high value may make the algorithm yield too much to competing flows.
- **kEnableConsecutiveFastStart** : Makes it possible to quickly reclaim free bandwidth, recommended to be enabled (true)
- **kEnablePacketPacing**: Paces out the RTP packets more smoothly and thus avoids issues with coalescing (a.k.a ACK compression)., recommended to be enabled (true)

ScreamTx METHODS

- ScreamTx(…) : Constructor. Allows to set alternative values to lossBeta, queueDelayTargetMin and enableSbd.
- registerNewStream(…) : Register a new stream with a given SSRC. A min and max bitrate is provided for the rate control. Furthermore a priority indicates priority relative to other streams. In addition it is possible to set alternative values for the rampUpSpeed, maxRtpQueueDelay, txQueueSizeFactor, queueDelayGuard and lossEventRateScale parameters.

ScreamTx METHODS

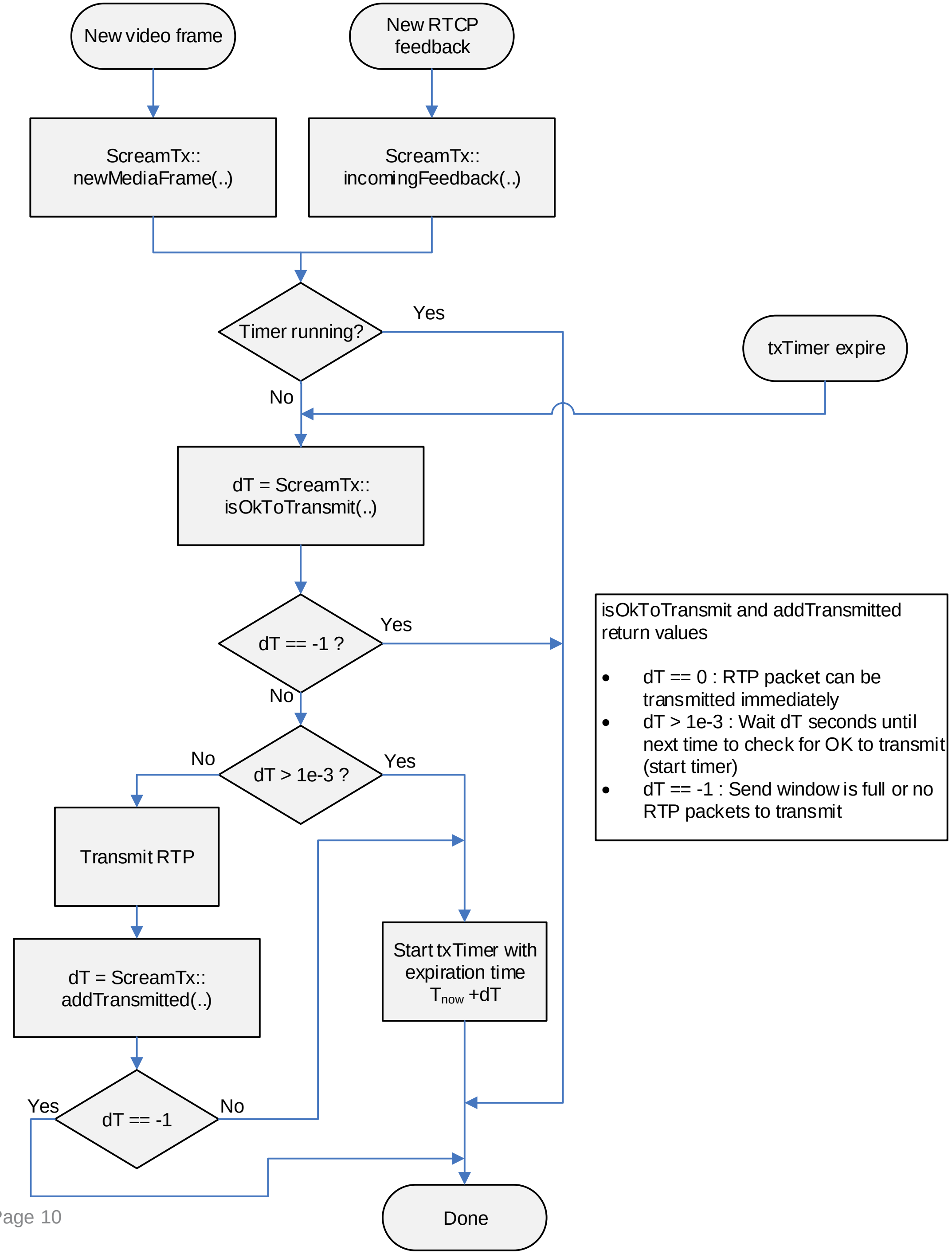
- newMediaFrame(…) : Called for each new media frame which can generate one or more RTP packets
- isOkToTransmit(…) : Determines if it is OK to transmit an RTP packet for any of the streams. The return values are :
  - 0.0 : OK to transmit one RTP packet from RTP queue indicated by parameter ssrc.
  - > 0.0 : Call this function again after a period given by the return value, this implements the packet pacing
  - -1.0 : No RTP packet to transmit or transmission scheduling does not allow for packet transmission

SCREAMTX METHODS

- addTransmitted(..) : Called when an RTP packet with SSRC is transmitted, return value indicates when isOkToTransmit(..) can be called again.
- incomingFeedback(..) : Called for each received SCReAM RTCP feedback message
- getTargetBitrate(..) : Get target bitrate for stream identified by parameters ssrc. NOTE ! This function reports the target bitrate <u>including</u> RTP overhead, the reason is that SCReAM operates in RTP packets. A media encoder must thus subtract the approximate RTP overhead.

SCREAMTX

- Flowchart describes the SCReAM sender side call flow



SCREAM RX

- Implements SCReAM received side functionality and generates RTCP feedback elements
  - Generation of byte correct RTCP packets is not done
  - It is not sure that RTCP is used at all!, RTP header extensions may be more beneficial in some cases.
  - RFC3611 XR blocks for RTCP feedback (see later).

SCREAMRX METHODS

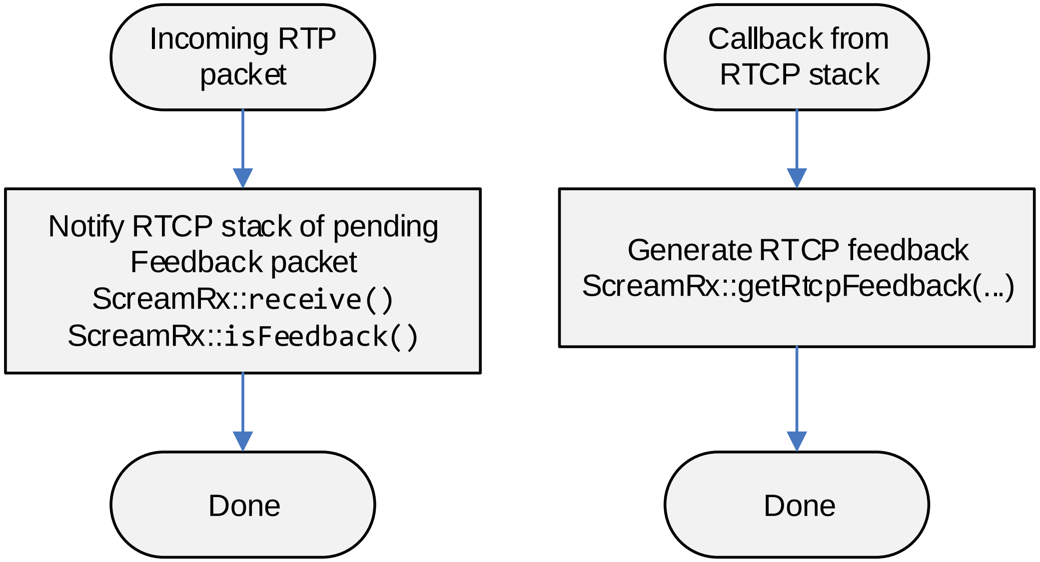
- receive(…) : Called each time an RTP packet is received, handling of new SSRC i.e. new streams is done automatically.
- isFeedback(…) : Returns true new RTP packets are received and SCReAM RTCP feedback can be transmitted

SCREAMRX METHODS

- getFeedback(…) : Get the feedback elements to generate a new SCReAM RTCP feedback packet. Function returns false if no stream with pending feedback available. The feedback elements are:
  - uint32_t ssrc : SSRC of sender
  - uint32_t receiveTimestamp : timestamp, default clock frequency = 1000Hz, important that both sender and received agree on the same clock frequency
  - uint16_t highestSeqNr : Highest received sequence number (possibly wrapped around)
  - uint64_t ackVector : 64 bit ACK vector that indicates reception of RTP packets prior to highestSeqNr. The most recent sequence numbers are indicated is the least significant bits, which means that if the ACK vector is truncated, then the LSB should be used.

ScreamRx

› Flow chart describes SCReAM received side call flow



FEEDBACK FORMAT
RFC3611 REALIZATION

- The feedback needed for basic functionality is
  - Timestamp
  - List of received SN, highest SN and ACK vector.
- Basic feedback can be implemented with RFC3611
  - Loss RLE report block
  - Packet Receipt Times block
- Feedback packet size : 44byte
  - assuming 4 RLE chunks

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|reserved |   PT=XR=207   |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SSRC                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     BT=2      | rsvd. |  T=0  |         block length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SSRC of source                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          begin_seq            |             end_seq           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          chunk 1              |             chunk 2           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                              ...                              :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          chunk n-1            |             chunk n           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     BT=3      | rsvd. |  T=0  |         block length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SSRC of source                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          begin_seq            |             end_seq           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Receipt time of packet begin_seq                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

FEEDBACK FORMAT
LOSS RLE REPORT BLOCK, RFC3611

- Loss RLE report block spans from begin_seq to end_seg
- 4 chunks should be sufficient

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     BT=2      | rsvd. |  T=0  |         block length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SSRC of source                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          begin_seq            |             end_seq           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          chunk 1              |             chunk 2           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                              ...                              :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          chunk n-1            |             chunk n           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

FEEDBACK FORMAT
PACKET RECEIPT TIMES BLOCK, RFC3611

- end_seq = (begin_seq+1) % 65536
- Receipt time stamp clock according to RFC3611 = Media RTP timestamp
  - It is however possible to use the default 1000Hz clock setting (kTimestampRate in ScreamTx.h)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     BT=3      | rsvd. |  T=0  |         block length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        SSRC of source                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          begin_seq            |             end_seq           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Receipt time of packet begin_seq                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

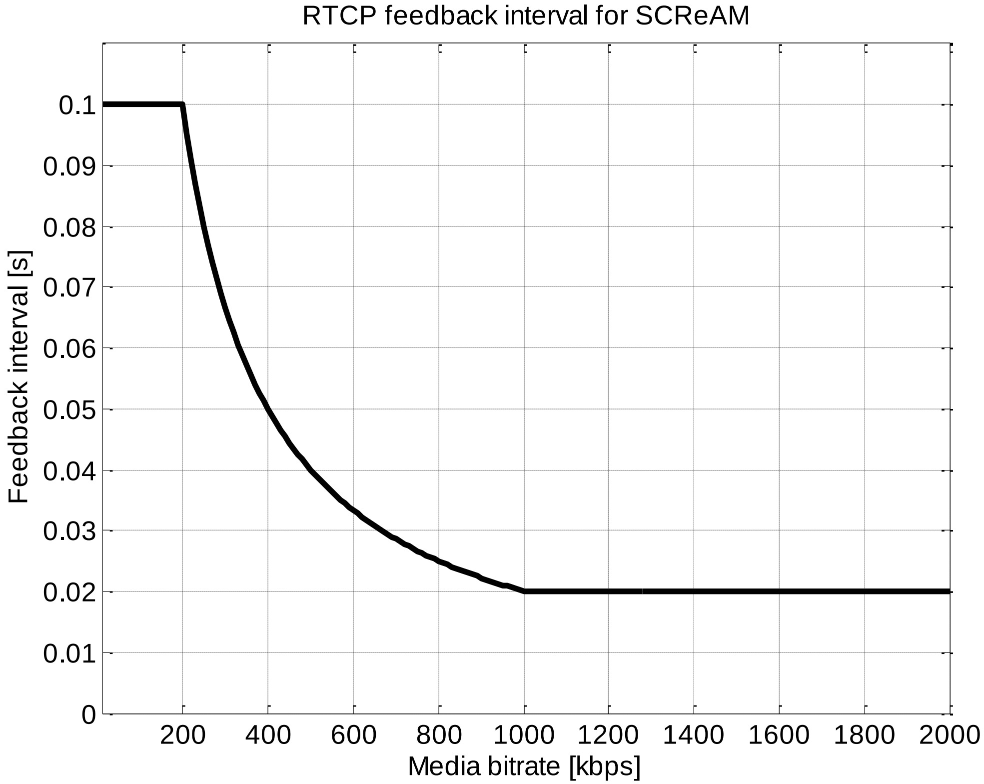
FALSE LOSS DETECTION

- Loss RLE block may not cover all received packets since last received feedback
  - Too low feedback rate
  - Feedback is lost
- Can lead to false loss detection
  - Should be a rare case but cannot be ignored

- Solution : Regular compound RTCP packets are expected to be transmitted at regular intervals (500ms), interval given by RFC4585 trr-int
  - cumulative number of packets lost : Can be used to undo a false loss detection

SIGNALING

- RFC4585 regular mode is sufficient
  - Early mode may be used for application layer signaling (ReTx, FIR..)
- Reduced size RTCP highly recommended
  - Regular Compound RTCP transmission given by trr-int
- Signaling rate dependent on bitrate
  - From 100-200ms at low bitrates to 10-20ms at high bitrates
  - Based on empirical data from experiments and simulation
- Expressed as a simple equation




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

ERICSSON