
# SCREAM EXPERIMENT PLATFORM

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.

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 in house in a state of 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.

References

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

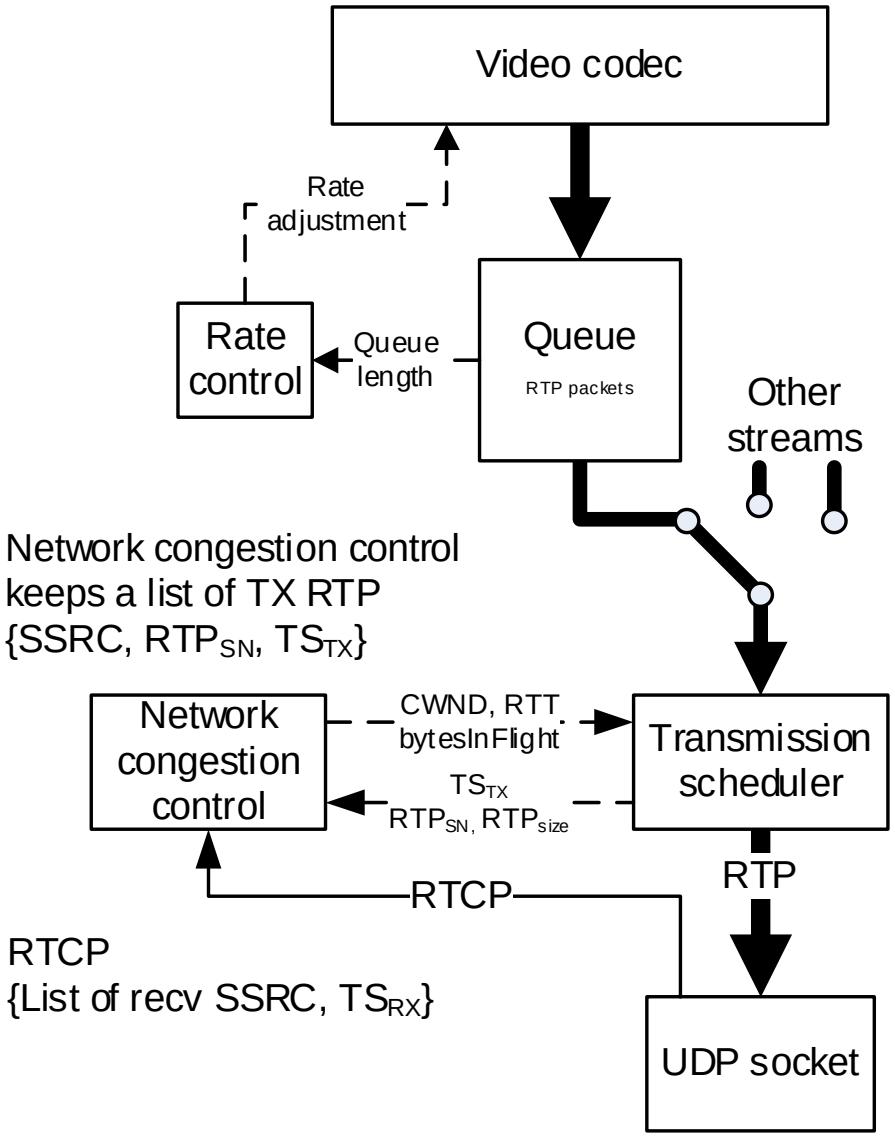
[2] Sigcomm paper http://dl.acm.org/citation.cfm?id=2631976

[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 controlled by transmission scheduler which is guided by the network congestion control



# SCREAM CLASSES

- Main SCReAM algorithm :
  - ScreamTx : SCReAM sender algorithm
  - ScreamRx : SCReAM receiver algorithm
- 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
- Time is counted in microseconds (guint64), 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.
- **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)
- **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.
- **kOwdGuard**: 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.

# ScreamTx methods

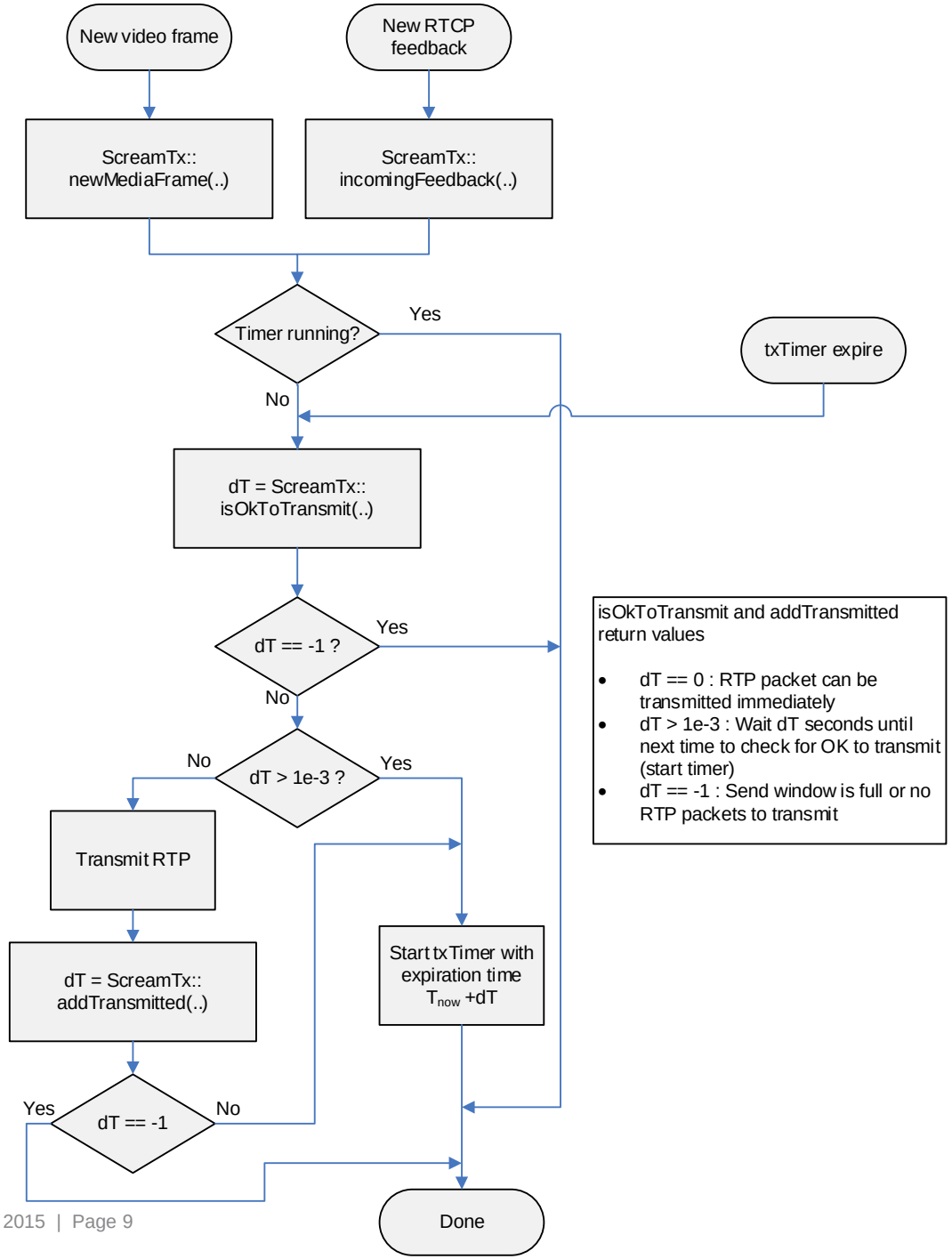
- registerNewStream(…) : Register a new stream with a given SSRC. A min and max bitrate and a framerate is provided for the rate control. Furthermore a priority indicates priority relative to other streams
- newMediaFrame(…) : Called for each new media frame which can generate one or more RTP packets
- 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 including 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



# ScreamRx

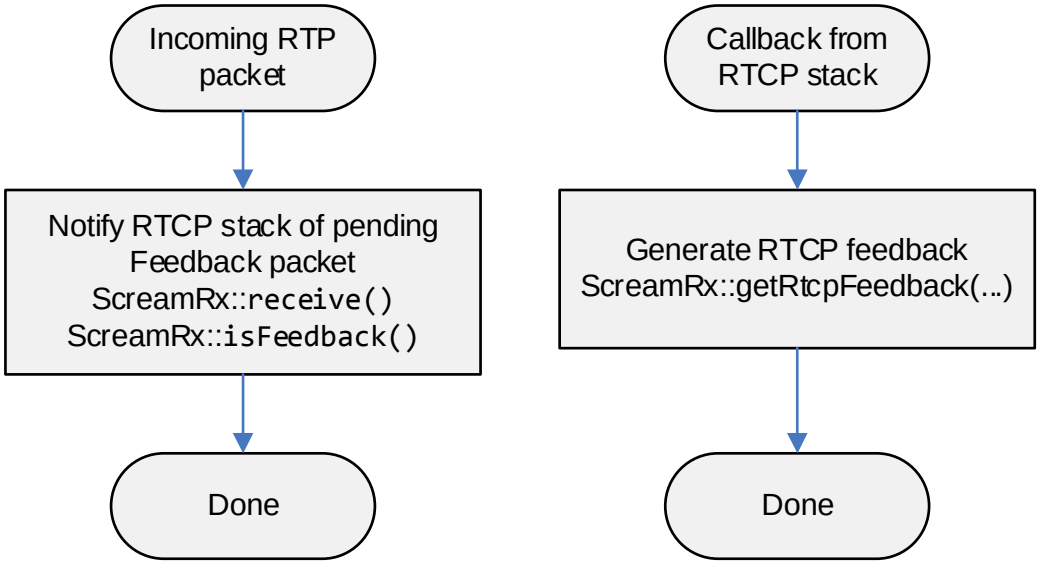
- Implements SCReAM received side functionality and generates RTCP feedback elements
  - Generation of byte correct RTCP packets is not done

# 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
- getFeedback(…) : Get the feedback elements to generate a new  SCReAM RTCP feedback packet. Function returns false if no stream with pending feedback available

# ScreamRx

- Flow chart describes SCReAM received side call flow



# COMPILING AND RUNNING

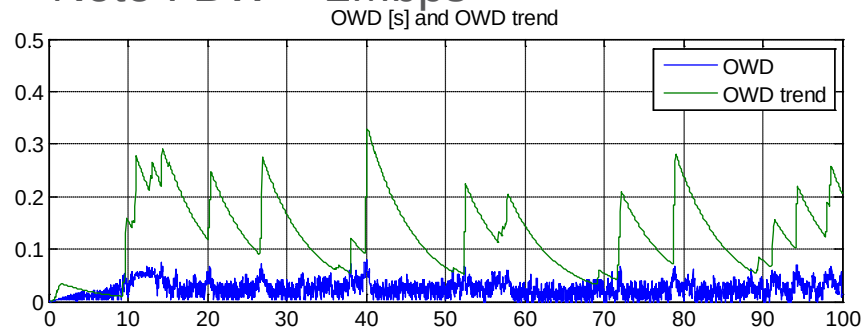
- Get the “all-in-one-bundle” from http://www.gtk.org/download/win32.php and unzip it
- Open up scream.sln in MS Visual Studio*
  - Set the C/C++ include paths in the two projects to
    - [GLIB]\lib\glib-2.0\include; [GLIB]\include\glib-2.0
  - Set additional library directories to
    - [GLIB]\lib
  - Make sure that glib-2.0.lib is set in the Linker additional dependencies
  - Ensure that GLib DLLs are accessible

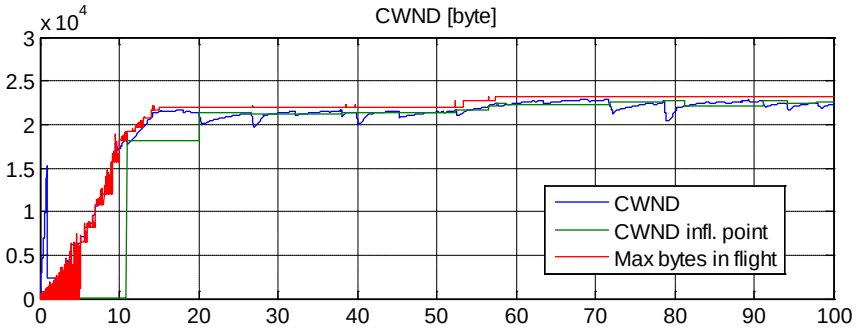
[GLIB] is the Glib installation path

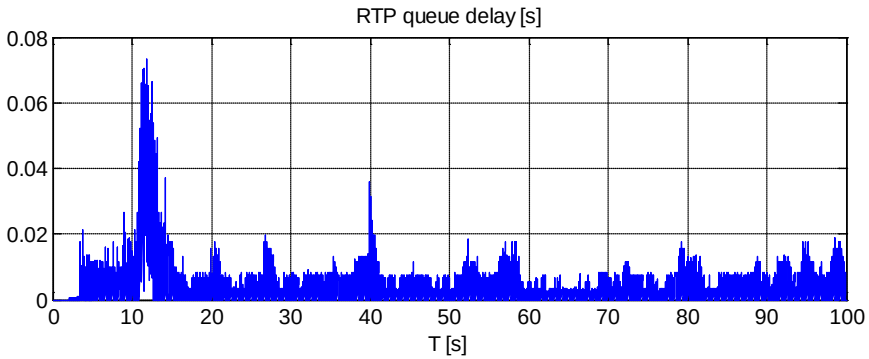
*Free community version from https://www.visualstudio.com/

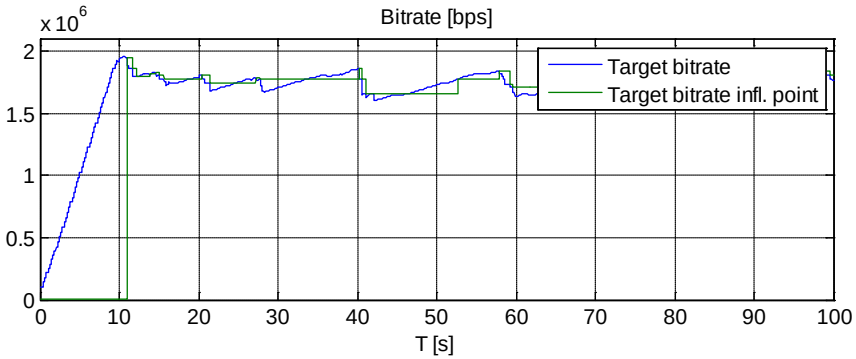
# EXAMPLE 1
# 1 VIDEO STREAM

- Build scream_01
- Open a command prompt, cd to the Debug folder
- **scream_01.exe > ../test.txt**
- In matlab execute
- **a = load('test.txt');test_01(a,50,2.1e6,3e4);**
- Note : BW = 2Mbps



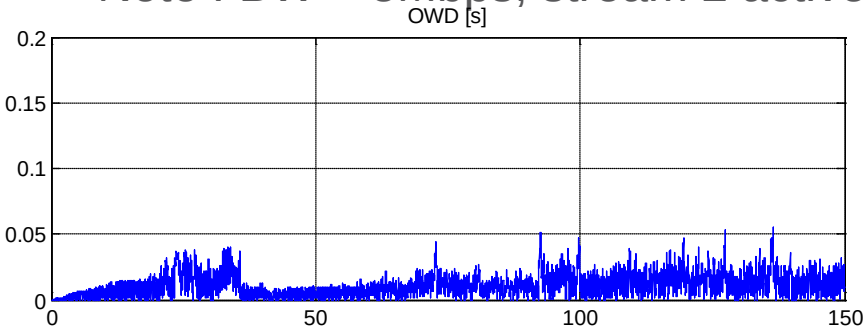



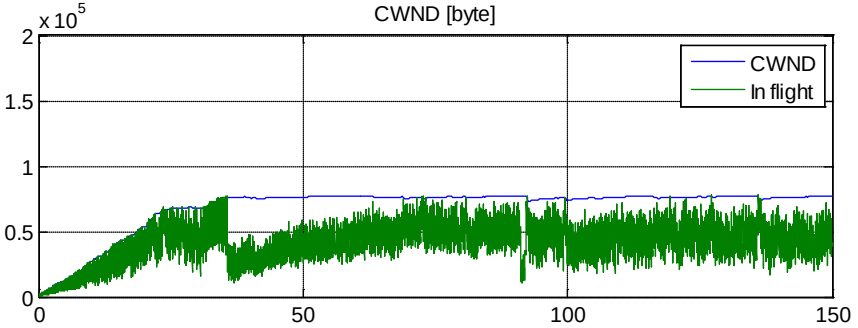



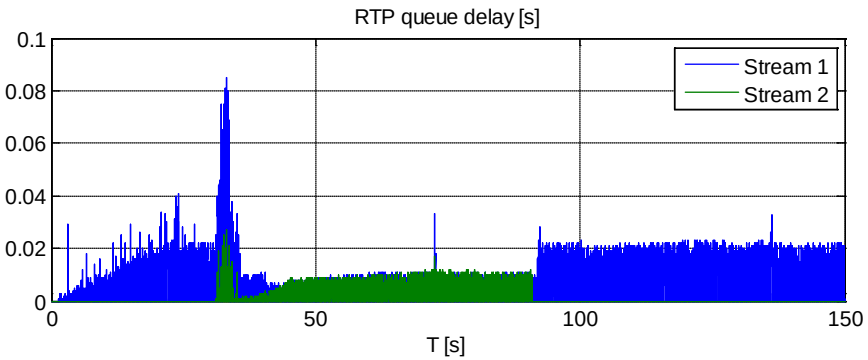



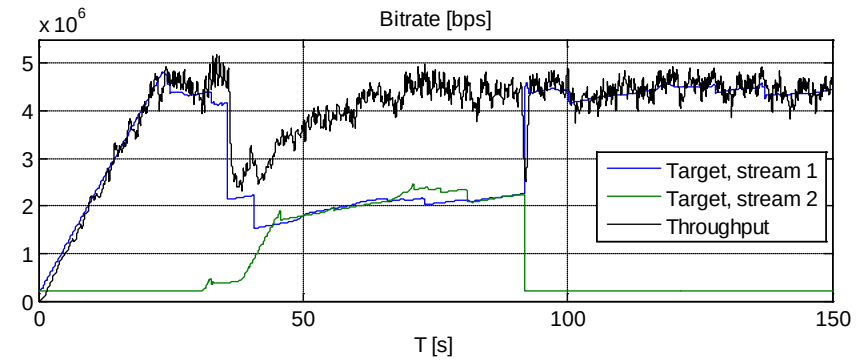
# EXAMPLE 2
# 2 VIDEO STREAMS

- Build scream_02
- Open a command prompt, cd to the Debug folder
- **scream_02.exe > ../test.txt**
- In matlab execute
- **a = load('test.txt');test_02(a,150,5.5e6,2e5);**
- Note : BW = 5Mbps, stream 2 active from T=31s to T=91s



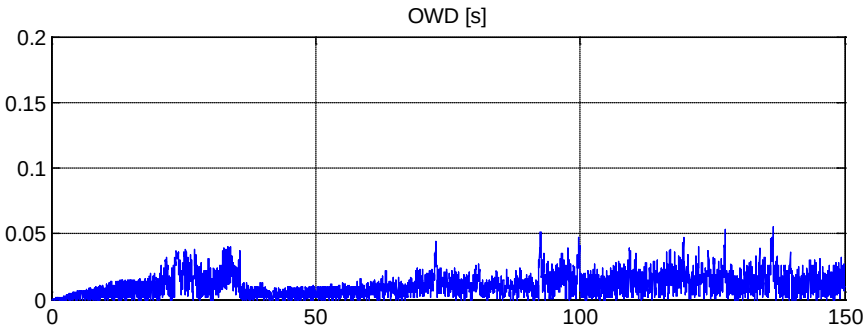



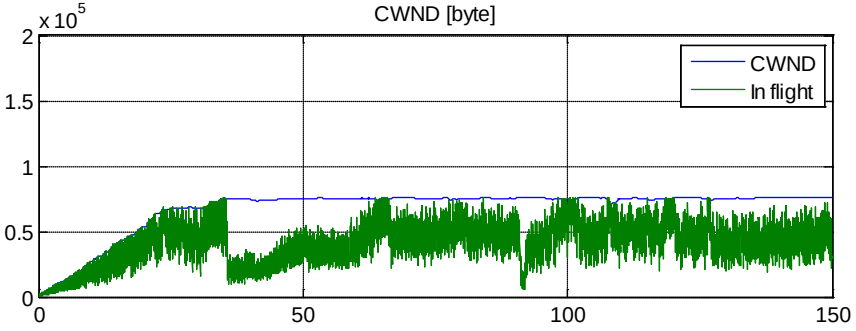



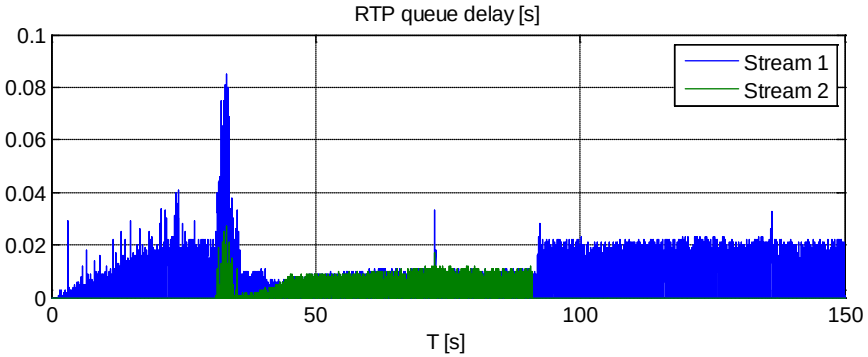



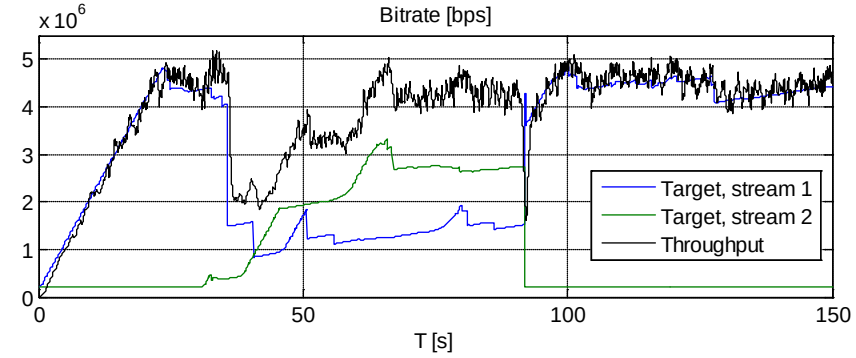
# EXAMPLE 2.1
# 2 VIDEO STREAMS

- Unequal priority
  - Stream 1 prio = 0.5
  - Stream 2 prio = 1.0



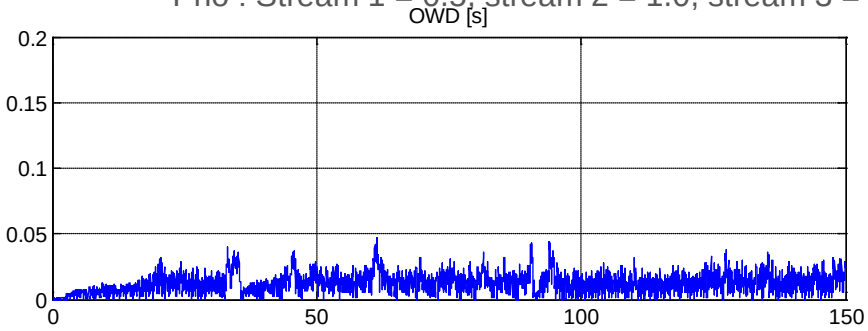



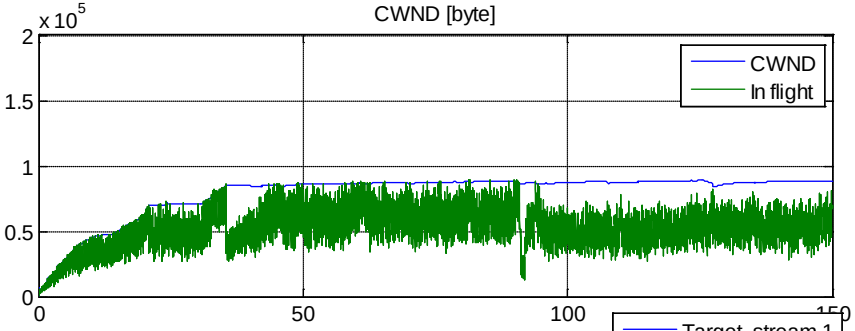



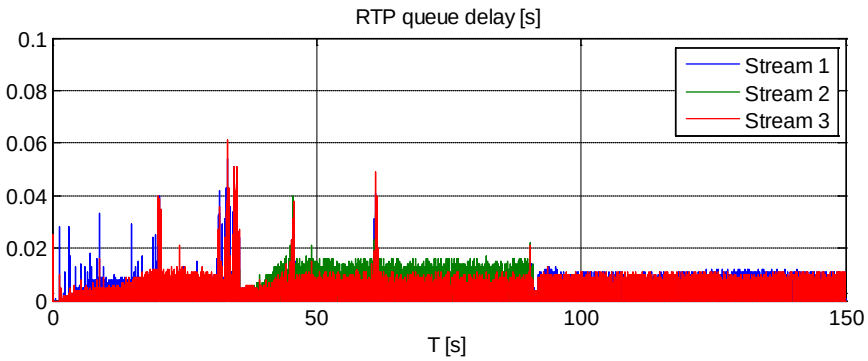



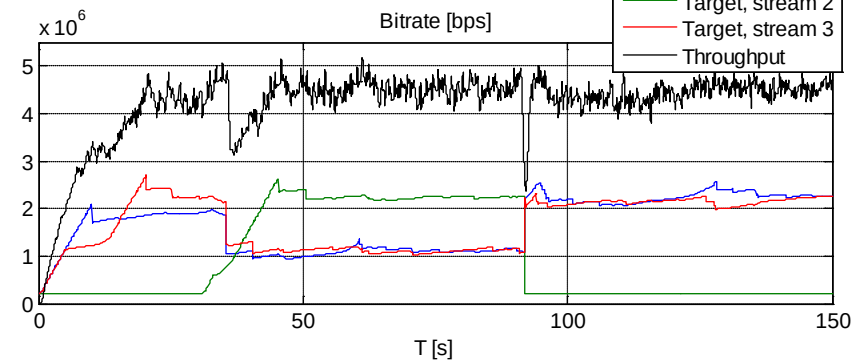
# EXAMPLE 2
# 3 VIDEO STREAMS

- Build scream_03
- Open a command prompt, cd to the Debug folder
- **scream_03.exe > ../test.txt**
- In matlab execute
- **a = load('test.txt');test_03(a,150,5.5e6,2e5);**
- Note : BW = 5Mbps, stream 2 active from T=31s to T=91s
  - Prio : Stream 1 = 0.5, stream 2 = 1.0, stream 3 = 0.5



ERICSSON