Abstract

This document describes a feedback message intended to enable congestion control for interactive real-time traffic. The RTP Media Congestion Avoidance Techniques (RMCAT) Working Group formed a design team to analyze feedback requirements from various congestion control algorithms and to design a generic feedback message to help ensure interoperability across those algorithms. The feedback message is designed for a sender-based congestion control, which means the receiver of the media will send necessary feedback to the sender of the media to perform the congestion control at the sender.
1. Introduction

For interactive real-time traffic the typical protocol choice is Realtime Transport Protocol (RTP) over User Datagram Protocol (UDP). RTP does not provide any guarantee of Quality of Service (QoS), reliable or timely delivery and expects the underlying transport protocol to do so. UDP alone certainly does not meet that expectation. However, RTP Control Protocol (RTCP) provides a mechanism to periodically send transport and media metrics to the media sender which can be utilized and extended for the purposes of RMCAT congestion control. For a congestion control algorithm which operates at the media sender, RTCP messages can be transmitted from the media receiver back to the media sender to enable congestion control. In the absence of standardized messages for this purpose, the congestion control algorithm designers have designed proprietary RTCP messages that convey only those parameters required for their respective designs. As a direct result, the different congestion control (a.k.a. rate adaptation) designs are not interoperable. To enable algorithm evolution as well as interoperability across designs (e.g., different rate adaptation algorithms), it is highly desirable to have generic congestion control feedback format.
To help achieve interoperability for unicast RTP congestion control, this memo proposes a common RTCP feedback format that can be used by NADA [I-D.ietf-rmcat-nada], SCReAM [I-D.ietf-rmcat-scream-cc], Google Congestion Control [I-D.ietf-rmcat-gcc] and Shared Bottleneck Detection [I-D.ietf-rmcat-sbd], and hopefully future RTP congestion control algorithms as well.

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].

In addition the terminology defined in [RFC3550], [RFC3551], [RFC3611], [RFC4585], and [RFC5506] applies.

3. Feedback Message

The design team analyzed the feedback requirements from the different proposed candidate in RMCAT WG. The analysis showed some commonalities between the proposed solution candidate and some can be derived from other information. The design team has agreed to have following packet information block in the feedback message to satisfy different requirement analyzed.

- **Packet Identifier**: RTP sequence number. The RTP packet header includes an incremental packet sequence number that the sender needs to correlate packets sent at the sender with packets received at the receiver.

- **Packet Arrival Time**: Arrival time stamp at the receiver of the media. The sender requires the arrival time stamp of the respective packet to determine delay and jitter the packet had experienced during transmission. In a sender based congestion control solution the sender requires to keep track of the sent packets - usually packet sequence number, packet size and packet send time. With the packet arrival time the sender can detect the delay and jitter information. Along with packet loss and delay information the sender can estimate the available bandwidth and thus adapt to the situation.

- **Packet Explicit Congestion Notification (ECN) Marking**: If ECN [RFC3168] is used, it is necessary to report on the 2-bit ECN mark in received packets, indicating for each packet whether it is marked not-ECT, ECT(0), ECT(1), or ECN-CE. If the path on which the media traffic traversing is ECN capable then the sender can use the Congestion Experienced (ECN-CE) marking information for congestion control. It is important that the receiver sends the
ECN-CE marking information of the packet back to the sender to take the advantages of ECN marking. Note that how the receiver gets the ECN marking information at application layer is out of the scope of this design team. Additional information for ECN use with RTP can be found at [RFC6679].

The feedback messages can have one or more of the above information blocks. For RTCP based feedback message the packet information block will be grouped by Synchronization Source (SSRC) identifier.

As a practical matter, we note that host Operating System (OS) process interruptions can occur at inopportune times. Thus, the recording of the sent times at the sender and arrival times at the receiver should be made with deliberate care. This is because the time duration of host OS interruptions can be significant relative to the precision desired in the one-way delay estimates. Specifically, the send time should be recorded at the latest opportunity prior to outputting the media packet at the sender (e.g., socket or RTP API) and the arrival time at the receiver (e.g., socket or RTP API) should be recorded at the earliest opportunity available to the receiver.

3.1. RTCP Congestion Control Feedback Report

Congestion control feedback can be sent as part of a regular scheduled RTCP report, or in an RTP/AVPF early feedback packet. If sent as early feedback, congestion control feedback MAY be sent in a non-compound RTCP packet [RFC5506] if the RTP/AVPF profile [RFC4585] or the RTP/SAVPF profile [RFC5124] is used.

Irrespective of how it is transported, the congestion control feedback is sent as a Transport Layer Feedback Message (RTCP packet type 205). The format of this RTCP packet is as follows:
The first 8 octets are the RTCP header, with PT=205 and FMT=CCFB specifying the remainder is a congestion control feedback packet, and including the SSRC of the packet sender. (NOTE TO RFC EDITOR: please replace CCFB here and in the above diagram with the IANA assigned RTCP feedback packet type)

Section 6.1 of [RFC4585] requires the RTCP header to be followed by the SSRC of the media source being reported upon. Accordingly, the RTCP header is followed by a report for each SSRC received, followed by the Report Timestamp.

The report for each SSRC received starts with the SSRC of that media source. Then, each sequence number between the begin_seq and end_seq (both inclusive) is represented by a packet metric block of 16-bits that contains the L, ECN, and ATO fields. If an odd number of reports are included, i.e., end_seq - begin_seq is odd then 16 bits of zero padding MUST be added after the last report, to align the RTCP packet to a four (4) bytes boundary. The L, ECN, and ATO fields are as follows:
Internet-Draft     Congestion Control Feedback in RTCP      October 2017

- L (1 bit): is a boolean to indicate if the packet was received. 0
  represents that the packet was not yet received and all the
  subsequent bits (ECN and ATO) are also set to 0. 1 represent the
  packet was received and the subsequent bits in the block need to
  be parsed.

- ECN (2 bits): is the echoed ECN mark of the packet. These are set
to 00 if not received, or if ECN is not used.

- Arrival time offset (ATO, 13 bits): it the relative arrival time
  of the RTP packets at the receiver before this feedback report was
  generated measured in milliseconds. It is calculated by
  subtracting the reception timestamp of the RTP packet denoted by
  this 16bit block and the timestamp (RTS) of this report. If the
  measured value is greater than 8.189 seconds (the value that would
  be coded as 0x1FFD), the value 0x1FFE MUST be reported to indicate
  an over-range positive measurement. If the measurement is
  unavailable, the value 0x1FFF MUST be reported.

Report Timestamp (RTS, 32 bits): represents the timestamp when this
report was generated. The sender of the feedback message decides on
the wall-clock. Usually, it should be derived from the same wall-
clock that is used for timestamping RTP packets arrival. Consistency
in the unit and resolution (10th of millisecond should be good enough
) is important here. In addition, the media sender can ask for a
specific resolution it wants.

4. Feedback Frequency and Overhead

There is a trade-off between speed and accuracy of reporting, and the
overhead of the reports. [I-D.ietf-rmcat-rtp-cc-feedback] discusses
this trade-off, and the possible rates of feedback.

It is a general understanding that the congestion control algorithms
will work better with more frequent feedback - per packet feedback.
However, RTCP bandwidth and transmission rules put some upper limits
on how frequently the RTCP feedback messages can be send from the
media receiver to the media sender. It has been shown
[I-D.ietf-rmcat-rtp-cc-feedback] that in most cases a per frame
feedback is a reasonable assumption on how frequent the RTCP feedback
messages can be transmitted. The design team also have noted that
even if a higher frequency of feedback is desired it is not viable if
the feedback messages starts to compete against the media traffic on
the feedback path during congestion period. Analyzing the feedback
interval requirement [feedback-requirements] it can be seen that the
candidate algorithms can perform with a feedback interval range of
50-200ms. A value within this range need to be negotiated at session
setup.

5. Design Rationale

The primary function of RTCP Sender Report (SR) / Receiver Report (RR) is to report the reception quality of media. The regular SR / RR reports contain information about observed jitter, fractional packet loss and cumulative packet loss. The original intent of this information was to assist flow and congestion control mechanisms. Even though it is possible to do congestion control based on information provided in the SR/RR reports it is not sufficient to design an efficient congestion control algorithm for interactive real-time communication. An efficient congestion control algorithm requires more fine grain information on per packet (see Section 3) to react to the congestion or to avoid funder congestion on the path.

Codec Control Message for AVPF [RFC5104] defines Temporary Maximum Media Bit Rate (TMMBR) message which conveys a temporary maximum bitrate limitation from the receiver of the media to the sender of the media. Even though it is not designed to replace congestion control, TMMBR has been used as a means to do receiver based congestion control where the session bandwidth is high enough to send frequent TMMBR messages especially with reduced sized reports [RFC5506]. This requires the receiver of the media to analyze the data reception, detect congestion level and recommend a maximum bitrate suitable for current available bandwidth on the path with an assumption that the sender of the media always honors the TMMBR message. This requirement is completely opposite of the sender based congestion control approach. Hence, TMMBR cannot be as a signaling means for a sender based congestion control mechanism. However, TMMBR should be viewed a complimentary signaling mechanism to establish receiver’s current view of acceptable maximum bitrate which a sender based congestion control should honor.

There are number of RTCP eXtended Report (XR) blocks have been defined for reporting of delay, loss and ECN marking. It is possible to combine several XR blocks to report the loss and ECN marking at the cost of overhead and complexity. However, there is no existing RTCP XR block to report packet arrival time.

Considering the issues discussed here it is rational to design a new congestion control feedback signaling mechanism for sender based congestion control algorithm.

6. Acknowledgements

This document is an outcome of RMCAT design team discussion. We would like to thank all participants specially Xiaoquing Zhu, Stefan Holmer, David, Ingemar Johansson and Randell Jesup for their valuable contribution to the discussions and to the document.
7. IANA Considerations

IANA is requested to assign a new value in the "FMT Values for RTPFB Payload Types" registry for the CCFB transport layer feedback packet described in Section 3.1.

8. Security Considerations

There is a risk of causing congestion if an on-path attacker modifies the feedback messages in such a manner to make available bandwidth greater than it is in reality. [More on security consideration TBD.]

9. References

9.1. Normative References

[I-D.ietf-rmcat-rtp-cc-feedback]


9.2. Informative References

[feedback-requirements]
"RMCAT Feedback Requirements",

[I-D.ietf-rmcat-gcc]

[I-D.ietf-rmcat-nada]

[I-D.ietf-rmcat-sbd]


Authors’ Addresses

Zaheduzzaman Sarker
Ericsson AB
Luleåe
Sweden

Phone: +46107173743
Email: zaheduzzaman.sarker@ericsson.com

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow  G12 8QQ
United Kingdom

Email: csp@csperkins.org

Varun Singh
Nemu Dialogue Systems Oy
Runeberginkatu 4c A 4
Helsinki 00100
Finland

Email: varun.singh@iki.fi
URI: http://www.callstats.io/

Michael A. Ramalho
Cisco Systems, Inc.
6310 Watercrest Way Unit 203
Lakewood Ranch, FL 34202
USA

Phone: +1 919 476 2038
Email: mramalho@cisco.com
Abstract

This document describes an RTCP feedback message intended to enable congestion control for interactive real-time traffic using RTP. The feedback message is designed for use with a sender-based congestion control algorithm, in which the receiver of an RTP flow sends RTCP feedback packets to the sender containing the information the sender needs to perform congestion control.

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 https://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 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of
1. Introduction

For interactive real-time traffic, such as video conferencing flows, the typical protocol choice is the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP). RTP does not provide any guarantee of Quality of Service (QoS), reliability, or timely delivery, and expects the underlying transport protocol to do so. UDP alone certainly does not meet that expectation. However, the RTP Control Protocol (RTCP) provides a mechanism by which the receiver of an RTP flow can periodically send transport and media quality metrics to the sender of that RTP flow. This information can be used by the sender to perform congestion control. In the absence of standardized messages for this purpose, designers of congestion control algorithms have developed proprietary RTCP messages that convey only those parameters needed for their respective designs. As a direct result, the different congestion control (i.e., rate adaptation) designs are not interoperable. To enable algorithm evolution as well as interoperability across designs (e.g., different rate adaptation algorithms), it is highly desirable to have generic congestion control feedback format.
To help achieve interoperability for unicast RTP congestion control, this memo proposes a common RTCP feedback packet format that can be used by NADA [I-D.ietf-rmcat-nada], SCReAM [RFC8298], Google Congestion Control [I-D.ietf-rmcat-gcc] and Shared Bottleneck Detection [RFC8382], and hopefully also by future RTP congestion control algorithms.

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].

In addition the terminology defined in [RFC3550], [RFC3551], [RFC3611], [RFC4585], and [RFC5506] applies.

3. RTCP Feedback for Congestion Control

Based on an analysis of NADA [I-D.ietf-rmcat-nada], SCReAM [RFC8298], Google Congestion Control [I-D.ietf-rmcat-gcc] and Shared Bottleneck Detection [RFC8382], the following per-RTP packet congestion control feedback information has been determined to be necessary:

- RTP sequence number: The receiver of an RTP flow needs to feedback the sequence numbers of the received RTP packets to the sender, so the sender can determine which packets were received and which were lost. Packet loss is used as an indication of congestion by many congestion control algorithms.

- Packet Arrival Time: The receiver of an RTP flow needs to feedback the arrival time of each RTP packet to the sender. Packet delay and/or delay variation (jitter) is used as a congestion signal by some congestion control algorithms.

- Packet Explicit Congestion Notification (ECN) Marking: If ECN [RFC3168], [RFC6679] is used, it is necessary to feedback the 2-bit ECN mark in received RTP packets, indicating for each RTP packet whether it is marked not-ECT, ECT(0), ECT(1), or ECN-CE. If the path used by the RTP traffic is ECN capable the sender can use Congestion Experienced (ECN-CE) marking information as a congestion control signal.

Every RTP flow is identified by its Synchronization Source (SSRC) identifier. Accordingly, the RTCP feedback format needs to group its reports by SSRC, sending one report block per received SSRC.

As a practical matter, we note that host operating system (OS) process interruptions can occur at inopportune times. Accordingly,
recording RTP packet send times at the sender, and the corresponding RTP packet arrival times at the receiver, needs to be done with deliberate care. This is because the time duration of host OS interruptions can be significant relative to the precision desired in the one-way delay estimates. Specifically, the send time needs to be recorded at the last opportunity prior to transmitting the RTP packet at the sender, and the arrival time at the receiver needs to be recorded at the earliest available opportunity.

3.1. RTCP Congestion Control Feedback Report

Congestion control feedback can be sent as part of a regular scheduled RTCP report, or in an RTP/AVPF early feedback packet. If sent as early feedback, congestion control feedback MAY be sent in a non-compound RTCP packet [RFC5506] if the RTP/AVPF profile [RFC4585] or the RTP/SAVPF profile [RFC5124] is used.

Irrespective of how it is transported, the congestion control feedback is sent as a Transport Layer Feedback Message (RTCP packet type 205). The format of this RTCP packet is shown in Figure 1:
Figure 1: RTCP Congestion Control Feedback Packet Format

The first eight octets comprise a standard RTCP header, with PT=205 and FMT=CCFB indicating that this is a congestion control feedback packet, and with the SSRC set to that of the sender of the RTCP packet. (NOTE TO RFC EDITOR: please replace CCFB here and in the above diagram with the IANA assigned RTCP feedback packet type, and remove this note)

Section 6.1 of [RFC4585] requires the RTCP header to be followed by the SSRC of the RTP flow being reported upon. Accordingly, the RTCP header is followed by a report block for each SSRC from which RTP packets have been received, followed by a Report Timestamp.

Each report block begins with the SSRC of the received RTP Stream on which it is reporting. Following this, the report block contains a 16-bit packet metric block for each RTP packet with sequence number in the range begin_seq to begin_seq+num_reports inclusive (calculated using arithmetic modulo 65536 to account for possible sequence number wrap-around). If the number of 16-bit packet metric blocks included
in the report block is not a multiple of two, then 16 bits of zero padding MUST be added after the last packet metric block, to align the end of the packet metric blocks with the next 32 bit boundary. The value of num_reports MAY be zero, indicating that there are no packet metric blocks included for that SSRC. Each report block MUST NOT include more than 16384 packet metric blocks (i.e., it MUST NOT report on more than one quarter of the sequence number space in a single report).

The contents of each 16-bit packet metric block comprises the L, ECN, and ATO fields as follows:

- **L (1 bit)**: is a boolean to indicate if the packet was received. 0 represents that the packet was not yet received and all the subsequent bits (ECN and ATO) are also set to 0. 1 represents the packet was received and the subsequent bits in the block need to be parsed.

- **ECN (2 bits)**: is the echoed ECN mark of the packet. These are set to 00 if not received, or if ECN is not used.

- **Arrival time offset (ATO, 13 bits)**: is the arrival time of the RTP packet at the receiver, as an offset before the time represented by the RTS field of this RTCP congestion control feedback report. The ATO field is in units of $1/1024$ seconds (this unit is chosen to give exact offsets from the RTS field) so, for example, an ATO value of 512 indicates that the corresponding RTP packet arrived exactly half a second before the time instant represented by the RTS field. If the measured value is greater than $8189/1024$ seconds (the value that would be coded as 0x1FFD), the value 0x1FFE MUST be reported to indicate an over-range measurement. If the measurement is unavailable, or if the arrival time of the RTP packet is after the time represented by the RTS field, then an ATO value of 0xFFFF MUST be reported for the packet.

The RTCP congestion control feedback report packet concludes with the Report Timestamp field (RTS, 32 bits). This denotes the time instant on which this packet is reporting, and is the instant from which the arrival time offset values are calculated. The value of RTS field is derived from the same clock used to generate the NTP timestamp field in RTCP Sender Report (SR) packets. It is formatted as the middle 32 bits of an NTP format timestamp, as described in Section 4 of [RFC3550].

RTCP congestion control feedback packets SHOULD include a report block for every active SSRC. The sequence number ranges reported in consecutive reports for a given SSRC will generally be contiguous, but overlapping reports MAY be sent (and need to be sent in cases
where RTP packet reordering occurs across the boundary between consecutive reports). If reports covering overlapping sequence number ranges are sent, information in later reports updates that in sent previous reports for RTP packets included in both reports. If an RTP packet was reported as received in one report, that packet MUST also be reported as received in any overlapping reports sent later that cover its sequence number range.

RTCP congestion control feedback packets can be large if they are sent infrequently relative to the number of RTP data packets. If an RTCP congestion control feedback packet is too large to fit within the path MTU, its sender SHOULD split it into multiple feedback packets. The RTCP reporting interval SHOULD be chosen such that feedback packets are sent often enough that they are small enough to fit within the path MTU ([I-D.ietf-rmcat-rtp-cc-feedback] provides guidance on how to choose the reporting interval).

If duplicate copies of a particular RTP packet are received, then the arrival time of the first copy to arrive MUST be reported. If any of the copies of the duplicated packet are ECN-CE marked, then an ECN-CE mark MUST be reported that for packet; otherwise the ECN mark of the first copy to arrive is reported.

If no packets are received from an SSRC in a reporting interval, a report block MAY be sent with begin_seq set to the highest sequence number previously received from that SSRC and num_reports set to zero (or, the report can simply to omitted). The corresponding SR/RR packet will have a non-increased extended highest sequence number received field that will inform the sender that no packets have been received, but it can ease processing to have that information available in the congestion control feedback reports too.

A report block indicating that certain RTP packets were lost is not to be interpreted as a request to retransmit the lost packets. The receiver of such a report might choose to retransmit such packets, provided a retransmission payload format has been negotiated, but there is no requirement that it do so.

4. Feedback Frequency and Overhead

There is a trade-off between speed and accuracy of reporting, and the overhead of the reports. [I-D.ietf-rmcat-rtp-cc-feedback] discusses this trade-off, suggests desirable RTCP feedback rates, and provides guidance on how to configure the RTCP bandwidth fraction, etc., to make appropriate use of the reporting block described in this memo. Specifications for RTP congestion control algorithms can also provide guidance.
It is a general understanding that the congestion control algorithms will work better with more frequent feedback — per packet feedback. However, RTCP bandwidth and transmission rules put some upper limits on how frequently the RTCP feedback messages can be send from the RTP receiver to the RTP sender. It has been shown [I-D.ietf-rmcat-rtp-cc-feedback] that in most cases a per frame feedback is a reasonable assumption on how frequent the RTCP feedback messages can be transmitted. It has also been noted that even if a higher frequency of feedback is desired it is not viable if the feedback messages starts to compete against the RTP traffic on the feedback path during congestion period. Analyzing the feedback interval requirement [feedback-requirements] it can be seen that the candidate algorithms can perform with a feedback interval range of 50-200ms. A value within this range need to be negotiated at session setup.

5. Response to Loss of Feedback Packets

Like all RTCP packets, RTCP congestion control feedback packets might be lost. All RTP congestion control algorithms MUST specify how they respond to the loss of feedback packets.

If only a single congestion control feedback packet is lost, an appropriate response is to assume that the level of congestion has remained roughly the same as the previous report. However, if multiple consecutive congestion control feedback packets are lost, the sender SHOULD rapidly reduce its sending rate towards zero, as this likely indicates a path failure. The RTP circuit breaker [RFC8083] provides further guidance.

6. SDP Signalling

A new "ack" feedback parameter, "ccfb", is defined for use with the "a=rtcp-fb:" SDP extension to indicate the use of the RTP Congestion Control feedback packet format defined in Section 3. The ABNF definition of this SDP parameter extension is:

```plaintext
rtcp-fb-ack-param = <See Section 4.2 of [RFC4585]>
rtcp-fb-ack-param =/ ccfb-par
ccfb-par = SP "ccfb"
```

When used with "ccfb" feedback, the wildcard payload type ("*")) MUST be used. This implies that the congestion control feedback is sent for all payload types in use in the session, including any FEC and retransmission payload types. An example of the resulting SDP attribute is:

```plaintext
a=rtcp-fb:* ack ccfb
```
The offer/answer rules for these SDP feedback parameters are specified in Section 4.2 of the RTP/AVPF profile [RFC4585].

An SDP offer might indicate support for both the congestion control feedback mechanism specified in this memo and one or more alternative congestion control feedback mechanisms that offer substantially the same semantics. In this case, the answering party SHOULD include only one of the offered congestion control feedback mechanisms in its answer. If a re-invite offering the same set of congestion control feedback mechanisms is received, the generated answer SHOULD choose the same congestion control feedback mechanism as in the original answer where possible.

When the SDP BUNDLE extension [I-D.ietf-mmusic-sdp-bundle-negotiation] is used for multiplexing, the "a=rtcp-fb:" attribute has multiplexing category IDENTICAL-PER-PT [I-D.ietf-mmusic-sdp-mux-attributes].

7. Relation to RFC 6679

Use of Explicit Congestion Notification (ECN) with RTP is described in [RFC6679]. That specifies how to negotiate the use of ECN with RTP, and defines an RTCP ECN Feedback Packet to carry ECN feedback reports. It uses an SDP "a=ecn-capable-rtp:" attribute to negotiate use of ECN, and the "a=rtcp-fb:" attributes with the "nack" parameter "ecn" to negotiate the use of RTCP ECN Feedback Packets.

The RTCP ECN Feedback Packet is not useful when ECN is used with the RTP Congestion Control Feedback Packet defined in this memo since it provides duplicate information. Accordingly, when congestion control feedback is to be used with RTP and ECN, the SDP offer generated MUST include an "a=ecn-capable-rtp:" attribute to negotiate ECN support, along with an "a=rtcp-fb:" attribute with the "ack" parameter "ccfb" to indicate that the RTP Congestion Control Feedback Packet is to be used for feedback. The "a=rtcp-fb:" attribute MUST NOT include the "nack" parameter "ecn", so the RTCP ECN Feedback Packet will not be used.

8. Design Rationale

The primary function of RTCP SR/RR packets is to report statistics on the reception of RTP packets. The reception report blocks sent in these packets contain information about observed jitter, fractional packet loss, and cumulative packet loss. It was intended that this information could be used to support congestion control algorithms, but experience has shown that it is not sufficient for that purpose. An efficient congestion control algorithm requires more fine grained
information on per packet reception quality than is provided by SR/RR packets to react effectively.

The Codec Control Messages for the RTP/AVPF profile [RFC5104] include a Temporary Maximum Media Bit Rate (TMMBR) message. This is used to convey a temporary maximum bit rate limitation from a receiver of RTP packets to their sender. Even though it was not designed to replace congestion control, TMMBR has been used as a means to do receiver based congestion control where the session bandwidth is high enough to send frequent TMMBR messages, especially when used with non-compound RTCP packets [RFC5506]. This approach requires the receiver of the RTP packets to monitor their reception, determine the level of congestion, and recommend a maximum bit rate suitable for current available bandwidth on the path; it also assumes that the RTP sender can/will respect that bit rate. This is the opposite of the sender based congestion control approach suggested in this memo, so TMMBR cannot be used to convey the information needed for a sender based congestion control. TMMBR could, however, be viewed a complementary mechanism that can inform the sender of the receiver’s current view of acceptable maximum bit rate.

A number of RTCP eXtended Report (XR) blocks have previously been defined to report details of packet loss, arrival times [RFC3611], delay [RFC6843], and ECN marking [RFC6679]. It is possible to combine several such XR blocks to report the detailed loss, arrival time, and ECN marking marking information needed for effective sender-based congestion control. However, the result has high overhead both in terms of bandwidth and complexity, due to the need to stack multiple reports.

Considering these issues, we believe it appropriate to design a new RTCP feedback mechanism to convey information for sender based congestion control algorithms. The new congestion control feedback RTCP packet described in Section 3 provides such a mechanism.

9. Acknowledgements

This document is based on the outcome of a design team discussion in the RTP Media Congestion Avoidance Techniques (RMCAT) working group. The authors would like to thank David Hayes, Stefan Holmer, Randell Jesup, Ingemar Johansson, Jonathan Lennox, Sergio Mena, Nils Ohlmeier, Magnus Westerlund, and Xiaoqing Zhu for their valuable feedback.
10. IANA Considerations

The IANA is requested to register one new RTP/AVPF Transport-Layer Feedback Message in the table for FMT values for RTPFB Payload Types [RFC4585] as defined in Section 3.1:

- Name: CCFB
- Long name: RTP Congestion Control Feedback
- Value: (to be assigned by IANA)
- Reference: (RFC number of this document, when published)

The IANA is also requested to register one new SDP "rtcp-fb" attribute "ack" parameter, "ccfb", in the SDP ("ack" and "nack" Attribute Values) registry:

- Value name: ccfb
- Long name: Congestion Control Feedback
- Usable with: ack
- Reference: (RFC number of this document, when published)

11. Security Considerations

The security considerations of the RTP specification [RFC3550], the applicable RTP profile (e.g., [RFC3551], [RFC3711], or [RFC4585]), and the RTP congestion control algorithm that is in use (e.g., [I-D.ietf-rmcat-nada], [RFC8298], [I-D.ietf-rmcat-gcc], or [RFC8382]) apply.

A receiver that intentionally generates inaccurate RTCP congestion control feedback reports might be able trick the sender into sending at a greater rate than the path can support, thereby congesting the path. This will negatively impact the quality of experience of that receiver. Since RTP is an unreliable transport, a sender can intentionally leave a gap in the RTP sequence number space without causing harm, to check that the receiver is correctly reporting losses.

An on-path attacker that can modify RTCP congestion control feedback packets can change the reports to trick the sender into sending at either an excessively high or excessively low rate, leading to denial of service. The secure RTCP profile [RFC3711] can be used to authenticate RTCP packets to protect against this attack.

12. References
12.1.  Normative References

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings,
"Negotiating Media Multiplexing Using the Session
Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-
negotiation-54 (work in progress), December 2018.

[I-D.ietf-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when
Multiplexing", draft-ietf-mmusic-sdp-mux-attributes-17
(work in progress), February 2018.

[I-D.ietf-rmcat-rtp-cc-feedback]
Perkins, C., "RTP Control Protocol (RTCP) Feedback for
Congestion Control in Interactive Multimedia Conferences",
draft-ietf-rmcat-rtp-cc-feedback-04 (work in progress),
July 2018.

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

of Explicit Congestion Notification (ECN) to IP",
RFC 3168, DOI 10.17487/RFC3168, September 2001,

[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,

[RFC3551]  Schulzrinne, H. and S. Casner, "RTP Profile for Audio and
Video Conferences with Minimal Control", STD 65, RFC 3551,
DOI 10.17487/RFC3551, July 2003,

"RTP Control Protocol Extended Reports (RTCP XR)",
RFC 3611, DOI 10.17487/RFC3611, November 2003,

[RFC3711]  Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K.
Norrman, "The Secure Real-time Transport Protocol (SRTP)",
RFC 3711, DOI 10.17487/RFC3711, March 2004,
<https://www.rfc-editor.org/info/rfc3711>.
12.2. Informative References

[feedback-requirements]
"RMCAT Feedback Requirements",

[I-D.ietf-rmcat-gcc]

[I-D.ietf-rmcat-nada]


Authors’ Addresses

Zaheduzzaman Sarker
Ericsson AB
Luleåe
Sweden

Phone: +46107173743
Email: zaheduzzaman.sarker@ericsson.com

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
United Kingdom

Email: csp@csperkins.org

Varun Singh
CALLSTATS I/O Oy
Annankatu 31-33 C 42
Helsinki 00100
Finland

Email: varun.singh@iki.fi
URI: http://www.callstats.io/
Abstract

This document describes NADA (network-assisted dynamic adaptation), a novel congestion control scheme for interactive real-time media applications, such as video conferencing. In the proposed scheme, the sender regulates its sending rate based on either implicit or explicit congestion signaling, in a unified approach. The scheme can benefit from explicit congestion notification (ECN) markings from network nodes. It also maintains consistent sender behavior in the absence of such markings, by reacting to queuing delays and packet losses instead.

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 https://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 March 8, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document.
publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Terminology ................................................. 3
3. System Overview ............................................. 3
4. Core Congestion Control Algorithm ......................... 5
   4.1. Mathematical Notations ................................. 5
   4.2. Receiver-Side Algorithm ................................. 8
   4.3. Sender-Side Algorithm .................................. 10
5. Practical Implementation of NADA ............................ 13
   5.1. Receiver-Side Operation ................................. 13
   5.1.1. Estimation of one-way delay and queuing delay ....... 13
   5.1.2. Estimation of packet loss/marking ratio ............. 13
   5.1.3. Estimation of receiving rate ........................ 14
   5.2. Sender-Side Operation .................................. 14
   5.2.1. Rate shaping buffer ................................ 15
   5.2.2. Adjusting video target rate and sending rate ....... 16
   5.3. Feedback Message Requirements .......................... 16
6. Discussions and Further Investigations ..................... 17
   6.1. Choice of delay metrics ................................. 17
   6.2. Method for delay, loss, and marking ratio estimation .. 18
   6.3. Impact of parameter values .............................. 18
   6.4. Sender-based vs. receiver-based calculation ............ 20
   6.5. Incremental deployment ................................ 20
7. Reference Implementations .................................... 20
8. Suggested Experiments ....................................... 21
9. IANA Considerations ......................................... 22
10. Security Considerations ..................................... 22
11. Acknowledgments ............................................ 22
12. Contributors ................................................ 22
13. References .................................................. 23
   13.1. Normative References ................................. 23
   13.2. Informative References ................................. 24
Appendix A. Network Node Operations ............................ 26
   A.1. Default behavior of drop tail queues ................... 27
   A.2. RED-based ECN marking ................................ 27
   A.3. Random Early Marking with Virtual Queues .............. 28
Authors’ Addresses ............................................ 28
1. Introduction

Interactive real-time media applications introduce a unique set of challenges for congestion control. Unlike TCP, the mechanism used for real-time media needs to adapt quickly to instantaneous bandwidth changes, accommodate fluctuations in the output of video encoder rate control, and cause low queuing delay over the network. An ideal scheme should also make effective use of all types of congestion signals, including packet loss, queuing delay, and explicit congestion notification (ECN) [RFC3168] markings. The requirements for the congestion control algorithm are outlined in [I-D.ietf-rmcat-cc-requirements]. It highlights that the desired congestion control scheme should avoid flow starvation and attain a reasonable fair share of bandwidth when competing against other flows, adapt quickly, and operate in a stable manner.

This document describes an experimental congestion control scheme called network-assisted dynamic adaptation (NADA). The design of NADA benefits from explicit congestion control signals (e.g., ECN markings) from the network, yet also operates when only implicit congestion indicators (delay and/or loss) are available. Such a unified sender behavior distinguishes NADA from other congestion control schemes for real-time media. In addition, its core congestion control algorithm is designed to guarantee stability for path round-trip-times (RTTs) below a prescribed bound (e.g., 250ms with default parameter choices). It further supports weighted bandwidth sharing among competing video flows with different priorities. The signaling mechanism consists of standard RTP timestamp [RFC3550] and RTCP feedback reports. The definition of the desired RTCP feedback message is described in detail in [I-D.ietf-avtc-core-cc-feedback-message] so as to support the successful operation of several congestion control schemes for real-time interactive media.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. System Overview

Figure 1 shows the end-to-end system for real-time media transport that NADA operates in. Note that there also exist network nodes along the reverse (potentially uncongested) path that the RTCP
feedback reports traverse. Those network nodes are not shown in the figure for sake of brevity.

![System Overview Diagram](image)

**Figure 1: System Overview**

- **Media encoder** with rate control capabilities. It encodes raw media (audio and video) frames into a compressed bitstream which is later packetized into RTP packets. As discussed in [RFC8593], the actual output rate from the encoder $r_{vout}$ may fluctuate around the target $r_{vin}$. Furthermore, it is possible that the encoder can only react to bit rate changes at rather coarse time intervals, e.g., once every 0.5 seconds.

- **RTP sender**: responsible for calculating the NADA reference rate based on network congestion indicators (delay, loss, or ECN marking reports from the receiver), for updating the video encoder with a new target rate $r_{vin}$, and for regulating the actual sending rate $r_{send}$ accordingly. The RTP sender also generates a sending timestamp for each outgoing packet.

- **RTP receiver**: responsible for measuring and estimating end-to-end delay (based on sender timestamp), packet loss (based on RTP sequence number), ECN marking ratios (based on [RFC6679]), and receiving rate ($r_{recv}$) of the flow. It calculates the aggregated congestion signal ($x_{curr}$) that accounts for queuing delay, ECN markings, and packet losses. The receiver also determines the mode for sender rate adaptation ($r_{mode}$) based on whether the flow has encountered any standing non-zero congestion. The receiver sends periodic RTCP reports back to the sender, containing values of $x_{curr}$, $r_{mode}$, and $r_{recv}$.

- **Network node** with several modes of operation. The system can work with the default behavior of a simple drop tail queue. It can also benefit from advanced AQM features such as PIE [RFC8033], FQ-CoDel [RFC8290], ECN marking based on RED [RFC7567], and PCN marking using a token bucket algorithm ([RFC6660]). Note that network node operation is out of control for the design of NADA.
4. Core Congestion Control Algorithm

Like TCP-Friendly Rate Control (TFRC)[Floyd-CCR00] [RFC5348], NADA is a rate-based congestion control algorithm. In its simplest form, the sender reacts to the collection of network congestion indicators in the form of an aggregated congestion signal, and operates in one of two modes:

- Accelerated ramp-up: when the bottleneck is deemed to be underutilized, the rate increases multiplicatively with respect to the rate of previously successful transmissions. The rate increase multiplier (gamma) is calculated based on observed round-trip-time and target feedback interval, so as to limit self-inflicted queuing delay.

- Gradual rate update: in the presence of non-zero aggregate congestion signal, the sending rate is adjusted in reaction to both its value (x_curr) and its change in value (x_diff).

This section introduces the list of mathematical notations and describes the core congestion control algorithm at the sender and receiver, respectively. Additional details on recommended practical implementations are described in Section 5.1 and Section 5.2.

4.1. Mathematical Notations

This section summarizes the list of variables and parameters used in the NADA algorithm. Figure 3 also includes the default values for choosing the algorithm parameters either to represent a typical setting in practical applications or based on theoretical and simulation studies. See Section 6.3 for some of the discussions on the impact of parameter values. Additional studies in real-world settings suggested in Section 8 could gather further insight on how to choose and adapt these parameter values in practical deployment.
### Notation | Variable Name
--- | ---
t\_curr | Current timestamp
t\_last | Last time sending/receiving a feedback
delta | Observed interval between current and previous feedback reports: delta = t\_curr-t\_last
r\_ref | Reference rate based on network congestion
r\_send | Sending rate
r\_recv | Receiving rate
r\_vin | Target rate for video encoder
r\_vout | Output rate from video encoder
d\_base | Estimated baseline delay
d\_fwd | Measured and filtered one-way delay
d\_queue | Estimated queuing delay
d\_tilde | Equivalent delay after non-linear warping
p\_mark | Estimated packet ECN marking ratio
p\_loss | Estimated packet loss ratio
x\_curr | Aggregate congestion signal
x\_prev | Previous value of aggregate congestion signal
x\_diff | Change in aggregate congestion signal w.r.t. its previous value: x\_diff = x\_curr - x\_prev
r\_mode | Rate update mode: (0 = accelerated ramp-up; 1 = gradual update)
gamma | Rate increase multiplier in accelerated ramp-up mode
loss\_int | Measured average loss interval in packet count
loss\_exp | Threshold value for setting the last observed packet loss to expiration
rtt | Estimated round-trip-time at sender
buffer\_len | Rate shaping buffer occupancy measured in bytes

**Figure 2: List of variables.**

### Notation | Parameter Name | Default Value
--- | --- | ---
PRIO | Weight of priority of the flow | 1.0
RMIN | Minimum rate of application supported by media encoder | 150Kbps
RMAX | Maximum rate of application supported by media encoder | 1.5Mbps
XREF | Reference congestion level | 10ms
KAPPA | Scaling parameter for gradual rate update calculation | 0.5
ETA | Scaling parameter for gradual rate update calculation | 2.0
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAU</td>
<td>Upper bound of RTT in gradual rate update calculation</td>
<td>500ms</td>
</tr>
<tr>
<td>DELTA</td>
<td>Target feedback interval</td>
<td>100ms</td>
</tr>
<tr>
<td>LOGWIN</td>
<td>Observation window in time for calculating packet summary statistics at receiver</td>
<td>500ms</td>
</tr>
<tr>
<td>QEPS</td>
<td>Threshold for determining queuing delay build up at receiver</td>
<td>10ms</td>
</tr>
<tr>
<td>DFILT</td>
<td>Bound on filtering delay</td>
<td>120ms</td>
</tr>
<tr>
<td>GAMMA_MAX</td>
<td>Upper bound on rate increase ratio for accelerated ramp-up</td>
<td>0.5</td>
</tr>
<tr>
<td>QBOUND</td>
<td>Upper bound on self-inflicted queuing delay during ramp up</td>
<td>50ms</td>
</tr>
<tr>
<td>MULTILOSS</td>
<td>Multiplier for self-scaling the expiration threshold of the last observed loss (loss_exp) based on measured average loss interval (loss_int)</td>
<td>7.0</td>
</tr>
<tr>
<td>QTH</td>
<td>Delay threshold for invoking non-linear warping</td>
<td>50ms</td>
</tr>
<tr>
<td>LAMBDA</td>
<td>Scaling parameter in the exponent of non-linear warping</td>
<td>0.5</td>
</tr>
<tr>
<td>PLRREF</td>
<td>Reference packet loss ratio</td>
<td>0.01</td>
</tr>
<tr>
<td>PMRREF</td>
<td>Reference packet marking ratio</td>
<td>0.01</td>
</tr>
<tr>
<td>DLOSS</td>
<td>Reference delay penalty for loss when packet loss ratio is at PLRREF</td>
<td>10ms</td>
</tr>
<tr>
<td>DMARK</td>
<td>Reference delay penalty for ECN marking when packet marking is at PMRREF</td>
<td>2ms</td>
</tr>
<tr>
<td>FPS</td>
<td>Frame rate of incoming video</td>
<td>30</td>
</tr>
<tr>
<td>BETA_S</td>
<td>Scaling parameter for modulating outgoing sending rate</td>
<td>0.1</td>
</tr>
<tr>
<td>BETA_V</td>
<td>Scaling parameter for modulating video encoder target rate</td>
<td>0.1</td>
</tr>
<tr>
<td>ALPHA</td>
<td>Smoothing factor in exponential smoothing of packet loss and marking ratios</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Figure 3: List of algorithm parameters and their default values.
4.2. Receiver-Side Algorithm

The receiver-side algorithm can be outlined as below:

On initialization:
set \(d_{\text{base}} = \infty\)
set \(p_{\text{loss}} = 0\)
set \(p_{\text{mark}} = 0\)
set \(r_{\text{recv}} = 0\)
set both \(t_{\text{last}}\) and \(t_{\text{curr}}\) as current time in milliseconds

On receiving a media packet:
obtain current timestamp \(t_{\text{curr}}\) from system clock
obtain from packet header sending time stamp \(t_{\text{sent}}\)
obtain one-way delay measurement: \(d_{\text{fwd}} = t_{\text{curr}} - t_{\text{sent}}\)
update baseline delay: \(d_{\text{base}} = \min(d_{\text{base}}, d_{\text{fwd}})\)
update queuing delay: \(d_{\text{queue}} = d_{\text{fwd}} - d_{\text{base}}\)
update packet loss ratio estimate \(p_{\text{loss}}\)
update packet marking ratio estimate \(p_{\text{mark}}\)
update measurement of receiving rate \(r_{\text{recv}}\)

On time to send a new feedback report \((t_{\text{curr}} - t_{\text{last}} > \text{DELTA})\):
calculate non-linear warping of delay \(d_{\tilde{\text{fwd}}}\) if packet loss exists
calculate current aggregate congestion signal \(x_{\text{curr}}\)
determine mode of rate adaptation for sender: \(r_{\text{mode}}\)
send feedback containing values of: \(r_{\text{mode}}, x_{\text{curr}},\) and \(r_{\text{recv}}\)
update \(t_{\text{last}}\) = \(t_{\text{curr}}\)

In order for a delay-based flow to hold its ground when competing against loss-based flows (e.g., loss-based TCP), it is important to distinguish between different levels of observed queuing delay. For instance, over wired connections, a moderate queuing delay value on the order of tens of milliseconds is likely self-inflicted or induced by other delay-based flows, whereas a high queuing delay value of several hundreds of milliseconds may indicate the presence of a loss-based flow that does not refrain from increased delay.

If the last observed packet loss is within the expiration window of \(\text{loss}_{\text{exp}}\) (measured in terms of packet counts), the estimated queuing delay follows a non-linear warping:
In (1), the queuing delay value is unchanged when it is below the first threshold $QTH$; otherwise it is scaled down following a non-linear curve. This non-linear warping is inspired by the delay-adaptive congestion window backoff policy in [Budzisz-TON11], so as to "gradually nudge" the controller to operate based on loss-induced congestion signals when competing against loss-based flows. The exact form of the non-linear function has been simplified with respect to [Budzisz-TON11]. The value of the threshold $QTH$ should be carefully tuned for different operational environments, so as to avoid potential risks of prematurely discounting the congestion signal level. Typically, a higher value of $QTH$ is required in a noisier environment (e.g., over wireless connections, or where the video stream encounters many time-varying background competing traffic) so as to stay robust against occasional non-congestion-induced delay spikes. Additional insights on how this value can be tuned or auto-tuned should be gathered from carrying out experimental studies in different real-world deployment scenarios.

The value of $loss_{exp}$ is configured to self-scale with the average packet loss interval $loss_{int}$ with a multiplier MULTILOSS:

$$loss_{exp} = MULTILOSS \times loss_{int}.$$  

Estimation of the average loss interval $loss_{int}$, in turn, follows Section 5.4 of the TCP Friendly Rate Control (TFRC) protocol [RFC5348].

In practice, it is recommended to linearly interpolate between the warped ($d_{tilde}$) and non-warped ($d_{queue}$) values of the queuing delay during the transitional period lasting for the duration of $loss_{int}$.

The aggregate congestion signal is:

$$x_{curr} = d_{tilde} + DMARK*p_{mark}^{\downarrow 2} + DLOSS*p_{loss}^{\downarrow 2} + DMARK*p_{mark}^{\downarrow 2} + DLOSS*p_{loss}^{\downarrow 2}.$$
Here, DMARK is prescribed reference delay penalty associated with ECN markings at the reference marking ratio of PMRREF; DLOSS is prescribed reference delay penalty associated with packet losses at the reference packet loss ratio of PLRREF. The value of DLOSS and DMARK does not depend on configurations at the network node. Since ECN-enabled active queue management schemes typically mark a packet before dropping it, the value of DLOSS SHOULD be higher than that of DMARK. Furthermore, the values of DLOSS and DMARK need to be set consistently across all NADA flows sharing the same bottleneck link, so that they can compete fairly.

In the absence of packet marking and losses, the value of $x_{\text{curr}}$ reduces to the observed queuing delay $d_{\text{queue}}$. In that case the NADA algorithm operates in the regime of delay-based adaptation.

Given observed per-packet delay and loss information, the receiver is also in a good position to determine whether the network is underutilized and recommend the corresponding rate adaptation mode for the sender. The criteria for operating in accelerated ramp-up mode are:

- No recent packet losses within the observation window LOGWIN; and
- No build-up of queuing delay: $d_{\text{fwd}} - d_{\text{base}} < QEPS$ for all previous delay samples within the observation window LOGWIN.

Otherwise the algorithm operates in graduate update mode.

4.3. Sender-Side Algorithm

The sender-side algorithm is outlined as follows:
on initialization:
set $r_{\text{ref}} = R_{\text{MIN}}$
set $\text{rtt} = 0$
set $x_{\text{prev}} = 0$
set $t_{\text{last}}$ and $t_{\text{curr}}$ as current system clock time

on receiving feedback report:
obtain current timestamp from system clock: $t_{\text{curr}}$
obtain values of rmode, $x_{\text{curr}}$, and $r_{\text{recv}}$ from feedback report
update estimation of rtt
measure feedback interval: $\text{delta} = t_{\text{curr}} - t_{\text{last}}$
if rmode == 0:
update $r_{\text{ref}}$ following accelerated ramp-up rules
else:
update $r_{\text{ref}}$ following gradual update rules
clip rate $r_{\text{ref}}$ within the range of minimum rate (RMIN) and maximum rate (RMAX).
$x_{\text{prev}} = x_{\text{curr}}$
$t_{\text{last}} = t_{\text{curr}}$

In accelerated ramp-up mode, the rate $r_{\text{ref}}$ is updated as follows:

\[
\text{gamma} = \min(GAMMA_{\text{MAX}}, \frac{QBOUND}{\text{rtt+DELTA+DFILT}}) \quad (3)
\]

\[
\begin{align*}
\text{r}_{\text{ref}} &= \max(r_{\text{ref}}, (1+\text{gamma}) \times r_{\text{recv}}) \\
&= \max(r_{\text{ref}}, (1+\text{gamma}) \times r_{\text{recv}}) \quad (4)
\end{align*}
\]

The rate increase multiplier gamma is calculated as a function of upper bound of self-inflicted queuing delay (QBOUND), round-trip-time (rtt), target feedback interval (DELTA) and bound on filtering delay for calculating d_queue (DFILT). It has a maximum value of GAMMA_MAX. The rationale behind (3)-(4) is that the longer it takes for the sender to observe self-inflicted queuing delay build-up, the more conservative the sender should be in increasing its rate, hence the smaller the rate increase multiplier.

In gradual update mode, the rate $r_{\text{ref}}$ is updated as:
\[ x_{\text{offset}} = x_{\text{curr}} - \text{PRIO} \times \text{XREF} \times \text{RMAX}/r_{\text{ref}} \quad (5) \]
\[ x_{\text{diff}} = x_{\text{curr}} - x_{\text{prev}} \quad (6) \]
\[
\frac{\Delta x_{\text{offset}}}{TAU} = r_{\text{ref}} - \frac{\text{KAPPA}}{TAU} \times \frac{x_{\text{offset}}}{TAU}
\]
\[
- \frac{\text{KAPPA}}{TAU} \times \frac{x_{\text{prev}}}{TAU}
\]
\[ r_{\text{ref}} = r_{\text{ref}} - \frac{\text{KAPPA}}{TAU} \times \frac{x_{\text{diff}}}{TAU} \quad (7) \]

The rate changes in proportion to the previous rate decision. It is affected by two terms: offset of the aggregate congestion signal from its value at equilibrium \((x_{\text{offset}})\) and its change \((x_{\text{diff}})\). Calculation of \(x_{\text{offset}}\) depends on maximum rate of the flow \((\text{RMAX})\), its weight of priority \((\text{PRIO})\), as well as a reference congestion signal \((\text{XREF})\). The value of \(\text{XREF}\) is chosen so that the maximum rate of \(\text{RMAX}\) can be achieved when the observed congestion signal level is below \(\text{PRIO} \times \text{XREF}\).

At equilibrium, the aggregated congestion signal stabilizes at \(x_{\text{curr}} = \text{PRIO} \times \text{XREF} \times \text{RMAX}/r_{\text{ref}}\). This ensures that when multiple flows share the same bottleneck and observe a common value of \(x_{\text{curr}}\), their rates at equilibrium will be proportional to their respective priority levels \((\text{PRIO})\) and the range between minimum and maximum rate. Values of the minimum rate \((\text{RMIN})\) and maximum rate \((\text{RMAX})\) will be provided by the media codec, for instance, as outlined by [I-D.ietf-rmcat-cc-codec-interactions]. In the absence of such information, NADA sender will choose a default value of 0 for \(\text{RMIN}\), and 3Mbps for \(\text{RMAX}\).

As mentioned in the sender-side algorithm, the final rate is always clipped within the dynamic range specified by the application:

\[ r_{\text{ref}} = \min(r_{\text{ref}}, \text{RMAX}) \quad (8) \]
\[ r_{\text{ref}} = \max(r_{\text{ref}}, \text{RMIN}) \quad (9) \]

The above operations ignore many practical issues such as clock synchronization between sender and receiver, filtering of noise in delay measurements, and base delay expiration. These will be addressed in Section 5.
5. Practical Implementation of NADA

5.1. Receiver-Side Operation

The receiver continuously monitors end-to-end per-packet statistics in terms of delay, loss, and/or ECN marking ratios. It then aggregates all forms of congestion indicators into the form of an equivalent delay and periodically reports this back to the sender. In addition, the receiver tracks the receiving rate of the flow and includes that in the feedback message.

5.1.1. Estimation of one-way delay and queuing delay

The delay estimation process in NADA follows a similar approach as in earlier delay-based congestion control schemes, such as LEDBAT [RFC6817]. For experimental implementations, instead of relying on RTP timestamps and the transmission time offset RTP header extension [RFC5450], the NADA sender can generate its own timestamp based on local system clock and embed that information in the transport packet header. The NADA receiver estimates the forward delay as having a constant base delay component plus a time varying queuing delay component. The base delay is estimated as the minimum value of one-way delay observed over a relatively long period (e.g., tens of minutes), whereas the individual queuing delay value is taken to be the difference between one-way delay and base delay. By re-estimating the base delay periodically, one can avoid the potential issue of base delay expiration, whereby an earlier measured base delay value is no longer valid due to underlying route changes or cumulative timing difference introduced by the clock rate skew between sender and receiver. All delay estimations are based on sender timestamps with a recommended granularity of 100 microseconds or finer.

The individual sample values of queuing delay should be further filtered against various non-congestion-induced noise, such as spikes due to processing "hiccup" at the network nodes. Therefore, in addition to calculating the value of queuing delay using \( d_{\text{queue}} = d_{\text{fwd}} - d_{\text{base}} \), as expressed in Section 5.1, current implementation further employs a minimum filter with a window size of 15 samples over per-packet queuing delay values.

5.1.2. Estimation of packet loss/marking ratio

The receiver detects packet losses via gaps in the RTP sequence numbers of received packets. For interactive real-time media application with stringent latency constraint (e.g., video conferencing), the receiver avoids the packet re-ordering delay by treating out-of-order packets as losses. The instantaneous packet
loss ratio \( p_{\text{inst}} \) is estimated as the ratio between the number of missing packets over the number of total transmitted packets within the recent observation window \( \text{LOGWIN} \). The packet loss ratio \( p_{\text{loss}} \) is obtained after exponential smoothing:

\[
p_{\text{loss}} = \text{ALPHA} \cdot p_{\text{inst}} + (1 - \text{ALPHA}) \cdot p_{\text{loss}}. \quad (10)
\]

The filtered result is reported back to the sender as the observed packet loss ratio \( p_{\text{loss}} \).

Estimation of packet marking ratio \( p_{\text{mark}} \) follows the same procedure as above. It is assumed that ECN marking information at the IP header can be passed to the receiving endpoint, e.g., by following the mechanism described in [RFC6679].

5.1.3. Estimation of receiving rate

It is fairly straightforward to estimate the receiving rate \( r_{\text{recv}} \). NADA maintains a recent observation window with time span of \( \text{LOGWIN} \), and simply divides the total size of packets arriving during that window over the time span. The receiving rate \( (r_{\text{recv}}) \) can be calculated at either the sender side based on the per-packet feedback from the receiver, or included as part of the feedback report.

5.2. Sender-Side Operation

Figure 4 provides a detailed view of the NADA sender. Upon receipt of an RTCP feedback report from the receiver, the NADA sender calculates the reference rate \( r_{\text{ref}} \) as specified in Section 4.3. It further adjusts both the target rate for the live video encoder \( r_{\text{vin}} \) and the sending rate \( r_{\text{send}} \) over the network based on the updated value of \( r_{\text{ref}} \) and rate shaping buffer occupancy \( \text{buffer_len} \).

The NADA sender behavior stays the same in the presence of all types of congestion indicators: delay, loss, and ECN marking. This unified approach allows a graceful transition of the scheme as the network shifts dynamically between light and heavy congestion levels.

5.2.1. Rate shaping buffer

The operation of the live video encoder is out of the scope of the design for the congestion control scheme in NADA. Instead, its behavior is treated as a black box.

A rate shaping buffer is employed to absorb any instantaneous mismatch between encoder rate output $r_{\text{vout}}$ and regulated sending rate $r_{\text{send}}$. Its current level of occupancy is measured in bytes and is denoted as $\text{buffer\_len}$.

A large rate shaping buffer contributes to higher end-to-end delay, which may harm the performance of real-time media communications. Therefore, the sender has a strong incentive to prevent the rate shaping buffer from building up. The mechanisms adopted are:

- To deplete the rate shaping buffer faster by increasing the sending rate $r_{\text{send}}$; and
- To limit incoming packets of the rate shaping buffer by reducing the video encoder target rate $r_{\text{vin}}$. 

---

Figure 4: NADA Sender Structure

---
5.2.2. Adjusting video target rate and sending rate

If the level of occupancy in the rate shaping buffer is accessible at the sender, such information can be leveraged to further adjust the target rate of the live video encoder $r_{vin}$ as well as the actual sending rate $r_{send}$. The purpose of such adjustments is to mitigate the additional latencies introduced by the rate shaping buffer. The amount of rate adjustment can be calculated as follows:

$$r_{diff_v} = \min(0.05 \times r_{ref}, BETA_V \times 8 \times buffer\_len \times FPS). \quad (11)$$

$$r_{diff_s} = \min(0.05 \times r_{ref}, BETA_S \times 8 \times buffer\_len \times FPS). \quad (12)$$

$$r_{vin} = \max(RMIN, r_{ref} - r_{diff_v}). \quad (13)$$

$$r_{send} = \min(RMAX, r_{ref} + r_{diff_s}). \quad (14)$$

In (11) and (12), the amount of adjustment is calculated as proportional to the size of the rate shaping buffer but is bounded by 5% of the reference rate $r_{ref}$ calculated from network congestion feedback alone. This ensures that the adjustment introduced by the rate shaping buffer will not counteract with the core congestion control process. Equations (13) and (14) indicate the influence of the rate shaping buffer. A large rate shaping buffer nudges the encoder target rate slightly below -- and the sending rate slightly above -- the reference rate $r_{ref}$. The final video target rate ($r_{vin}$) and sending rate ($r_{send}$) are further bounded within the original range of $[RMIN, RMAX]$.

Intuitively, the amount of extra rate offset needed to completely drain the rate shaping buffer within the duration of a single video frame is given by $8 \times buffer\_len \times FPS$, where FPS stands for the reference frame rate of the video. The scaling parameters $BETA_V$ and $BETA_S$ can be tuned to balance between the competing goals of maintaining a small rate shaping buffer and deviating from the reference rate point. Empirical observations show that the rate shaping buffer for a responsive live video encoder typically stays empty and only occasionally holds a large frame (e.g., when an intra-frame is produced) in transit. Therefore, the rate adjustment introduced by this mechanism is expected to be minor. For instance, a rate shaping buffer of 2000 Bytes will lead to a rate adjustment of 48Kbps given the recommended scaling parameters of $BETA_V = 0.1$ and $BETA_S = 0.1$ and reference frame rate of $FPS = 30$.

5.3. Feedback Message Requirements

The following list of information is required for NADA congestion control to function properly:
- Recommended rate adaptation mode (rmode): a 1-bit flag indicating whether the sender should operate in accelerated ramp-up mode (rmode=0) or gradual update mode (rmode=1).

- Aggregated congestion signal (x_curr): the most recently updated value, calculated by the receiver according to Section 4.2. This information can be expressed with a unit of 100 microsecond (i.e., 1/10 of a millisecond) in 15 bits. This allows a maximum value of x_curr at approximately 3.27 second.

- Receiving rate (r_recv): the most recently measured receiving rate according to Section 5.1.3. This information is expressed with a unit of bits per second (bps) in 32 bits (unsigned int). This allows a maximum rate of approximately 4.3Gbps, approximately 1000 times of the streaming rate of a typical high-definition (HD) video conferencing session today. This field can be expanded further by a few more bytes, in case an even higher rate need to be specified.

The above list of information can be accommodated by 48 bits, or 6 bytes, in total. They can be either included in the feedback report from the receiver, or, in the case where all receiver-side calculations are moved to the sender, derived from per-packet information from the feedback message as defined in [I-D.ietf-avtcore-cc-feedback-message]. Choice of the feedback message interval DELTA is discussed in Section 6.3. A target feedback interval of DELTA=100ms is recommended.

6. Discussions and Further Investigations

This section discussed the various design choices made by NADA, potential alternative variants of its implementation, and guidelines on how the key algorithm parameters can be chosen. Section 8 recommends additional experimental setups to further explore these topics.

6.1. Choice of delay metrics

The current design works with relative one-way-delay (OWD) as the main indication of congestion. The value of the relative OWD is obtained by maintaining the minimum value of observed OWD over a relatively long time horizon and subtract that out from the observed absolute OWD value. Such an approach cancels out the fixed difference between the sender and receiver clocks. It has been widely adopted by other delay-based congestion control approaches such as [RFC6817]. As discussed in [RFC6817], the time horizon for tracking the minimum OWD needs to be chosen with care: it must be long enough for an opportunity to observe the minimum OWD with zero
standing queue along the path, and sufficiently short so as to timely reflect "true" changes in minimum OWD introduced by route changes and other rare events and to mitigate the cumulative impact of clock rate skew over time.

The potential drawback in relying on relative OWD as the congestion signal is that when multiple flows share the same bottleneck, the flow arriving late at the network experiencing a non-empty queue may mistakenly consider the standing queuing delay as part of the fixed path propagation delay. This will lead to slightly unfair bandwidth sharing among the flows.

Alternatively, one could move the per-packet statistical handling to the sender instead and use relative round-trip-time (RTT) in lieu of relative OWD, assuming that per-packet acknowledgments are available. The main drawback of RTT-based approach is the noise in the measured delay in the reverse direction.

Note that the choice of either delay metric (relative OWD vs. RTT) involves no change in the proposed rate adaptation algorithm. Therefore, comparing the pros and cons regarding which delay metric to adopt can be kept as an orthogonal direction of investigation.

6.2. Method for delay, loss, and marking ratio estimation

Like other delay-based congestion control schemes, performance of NADA depends on the accuracy of its delay measurement and estimation module. Appendix A in [RFC6817] provides an extensive discussion on this aspect.

The current recommended practice of applying minimum filter with a window size of 15 samples suffices in guarding against processing delay outliers observed in wired connections. For wireless connections with a higher packet delay variation (PDV), more sophisticated techniques on de-noising, outlier rejection, and trend analysis may be needed.

More sophisticated methods in packet loss ratio calculation, such as that adopted by [Floyd-CCR00], will likely be beneficial. These alternatives are part of the experiments this document proposes.

6.3. Impact of parameter values

In the gradual rate update mode, the parameter TAU indicates the upper bound of round-trip-time (RTT) in feedback control loop. Typically, the observed feedback interval delta is close to the target feedback interval DELTA, and the relative ratio of delta/TAU versus ETA dictates the relative strength of influence from the
aggregate congestion signal offset term ($x_{offset}$) versus its recent change ($x_{diff}$), respectively. These two terms are analogous to the integral and proportional terms in a proportional-integral (PI) controller. The recommended choice of $TAU=500\text{ms}$, $DELTA=100\text{ms}$ and $ETA = 2.0$ corresponds to a relative ratio of 1:10 between the gains of the integral and proportional terms. Consequently, the rate adaptation is mostly driven by the change in the congestion signal with a long-term shift towards its equilibrium value driven by the offset term. Finally, the scaling parameter $KAPPA$ determines the overall speed of the adaptation and needs to strike a balance between responsiveness and stability.

The choice of the target feedback interval $DELTA$ needs to strike the right balance between timely feedback and low RTCP feedback message counts. A target feedback interval of $DELTA=100\text{ms}$ is recommended, corresponding to a feedback bandwidth of 16Kbps with 200 bytes per feedback message --- approximately 1.6% overhead for a 1Mbps flow. Furthermore, both simulation studies and frequency-domain analysis in [IETF-95] have established that a feedback interval below 250ms (i.e., more frequently than 4 feedback messages per second) will not break up the feedback control loop of NADA congestion control.

In calculating the non-linear warping of delay in (1), the current design uses fixed values of $QTH$ for determining whether to perform the non-linear warping. Its value should be carefully tuned for different operational environments (e.g., over wired vs. wireless connections), so as to avoid the potential risk of prematurely discounting the congestion signal level. It is possible to adapt its value based on past observed patterns of queuing delay in the presence of packet losses. It needs to be noted that the non-linear warping mechanism may lead to multiple NADA streams stuck in loss-based mode when competing against each other.

In calculating the aggregate congestion signal $x_{curr}$, the choice of $DMARK$ and $DLOSS$ influence the steady-state packet loss/marking ratio experienced by the flow at a given available bandwidth. Higher values of $DMARK$ and $DLOSS$ result in lower steady-state loss/marking ratios, but are more susceptible to the impact of individual packet loss/marking events. While the value of $DMARK$ and $DLOSS$ are fixed and predetermined in the current design, this document also encourages further explorations of a scheme for automatically tuning these values based on desired bandwidth sharing behavior in the presence of other competing loss-based flows (e.g., loss-based TCP).
6.4. Sender-based vs. receiver-based calculation

In the current design, the aggregated congestion signal $x_{curr}$ is calculated at the receiver, keeping the sender operation completely independent of the form of actual network congestion indications (delay, loss, or marking) in use.

Alternatively, one can shift receiver-side calculations to the sender, whereby the receiver simply reports on per-packet information via periodic feedback messages as defined in [I-D.ietf-avtcore-cc-feedback-message]. Such an approach enables interoperability amongst senders operating on different congestion control schemes, but requires slightly higher overhead in the feedback messages. See additional discussions in [I-D.ietf-avtcore-cc-feedback-message] regarding the desired format of the feedback messages and the recommended feedback intervals.

6.5. Incremental deployment

One nice property of NADA is the consistent video endpoint behavior irrespective of network node variations. This facilitates gradual, incremental adoption of the scheme.

Initially, the proposed congestion control mechanism can be implemented without any explicit support from the network, and relies solely on observed relative one-way delay measurements and packet loss ratios as implicit congestion signals.

When ECN is enabled at the network nodes with RED-based marking, the receiver can fold its observations of ECN markings into the calculation of the equivalent delay. The sender can react to these explicit congestion signals without any modification.

Ultimately, networks equipped with proactive marking based on token bucket level metering can reap the additional benefits of zero standing queues and lower end-to-end delay and work seamlessly with existing senders and receivers.

7. Reference Implementations

The NADA scheme has been implemented in both [ns-2] and [ns-3] simulation platforms. The implementation in ns-2 hosts the calculations as described in Section 4.2 at the receiver side, whereas the implementation in ns-3 hosts these receiver-side calculations at the sender for the sake of interoperability. Extensive ns-2 simulation evaluations of an earlier version of the draft are documented in [Zhu-PV13]. An open source implementation of NADA as part of a ns-3 module is available at [ns3-rmcat].
Evaluation results of the current draft based on ns-3 are presented in [IETF-90] and [IETF-91] for wired test cases as documented in [I-D.ietf-rmcat-eval-test]. Evaluation results of NADA over WiFi-based test cases as defined in [I-D.ietf-rmcat-wireless-tests] are presented in [IETF-93]. These simulation-based evaluations have shown that NADA flows can obtain their fair share of bandwidth when competing against each other. They typically adapt fast in reaction to the arrival and departure of other flows, and can sustain a reasonable throughput when competing against loss-based TCP flows.

[IETF-90] describes the implementation and evaluation of NADA in a lab setting. Preliminary evaluation results of NADA in single-flow and multi-flow test scenarios have been presented in [IETF-91].

A reference implementation of NADA has been carried out by modifying the WebRTC module embedded in the Mozilla open source browser. Presentations from [IETF-103] and [IETF-105] document real-world evaluations of the modified browser driven by NADA. The experimental setting involve remote connections with endpoints over either home or enterprise wireless networks. These evaluations validate the effectiveness of NADA flows in recovering quickly from throughput drops caused by intermittent delay spikes over the last-hop wireless connections.

8. Suggested Experiments

NADA has been extensively evaluated under various test scenarios, including the collection of test cases specified by [I-D.ietf-rmcat-eval-test] and the subset of WiFi-based test cases in [I-D.ietf-rmcat-wireless-tests]. Additional evaluations have been carried out to characterize how NADA interacts with various active queue management (AQM) schemes such as RED, CoDel, and PIE. Most of these evaluations have been carried out in simulators. A few key test cases have been evaluated in lab environments with implementations embedded in video conferencing clients. It is strongly recommended to carry out implementation and experimentation of NADA in real-world settings. Such exercise will provide insights on how to choose or automatically adapt the values of the key algorithm parameters (see list in Figure 3) as discussed in Section 6.

Additional experiments are suggested for the following scenarios and preferably over real-world networks:

- Experiments reflecting the setup of a typical WAN connection.

- Experiments with ECN marking capability turned on at the network for existing test cases.
Experiments with multiple NADA streams bearing different user-specified priorities.

Experiments with additional access technologies, especially over cellular networks such as 3G/LTE.

Experiments with various media source contents, including audio only, audio and video, and application content sharing (e.g., slide shows).

9. IANA Considerations

This document makes no request of IANA.

10. Security Considerations

The rate adaptation mechanism in NADA relies on feedback from the receiver. As such, it is vulnerable to attacks where feedback messages are hijacked, replaced, or intentionally injected with misleading information resulting in denial of service, similar to those that can affect TCP. It is therefore RECOMMENDED that the RTCP feedback message is at least integrity checked. In addition, [I-D.ietf-avtcore-cc-feedback-message] discusses the potential risk of a receiver providing misleading congestion feedback information and the mechanisms for mitigating such risks.

The modification of sending rate based on send-side rate shaping buffer may lead to temporary excessive congestion over the network in the presence of a unresponsive video encoder. However, this effect can be mitigated by limiting the amount of rate modification introduced by the rate shaping buffer, bounding the size of the rate shaping buffer at the sender, and maintaining a maximum allowed sending rate by NADA.

11. Acknowledgments

The authors would like to thank Randell Jesup, Luca De Cicco, Piers O’Hanlon, Ingemar Johansson, Stefan Holmer, Cesar Ilharco Magalhaes, Safiqul Islam, Michael Welzl, Mirja Kuhlewind, Karen Elisabeth Egede Nielsen, Julius Flohr, Roland Bless, Andreas Smas, and Martin Stiemerling for their valuable review comments and helpful input to this specification.

12. Contributors

The following individuals have contributed to the implementation and evaluation of the proposed scheme, and therefore have helped to validate and substantially improve this specification.
Paul E. Jones <paulej@packetizer.com> of Cisco Systems implemented an early version of the NADA congestion control scheme and helped with its lab-based testbed evaluations.

Jiantao Fu <jianfu@cisco.com> of Cisco Systems helped with the implementation and extensive evaluation of NADA both in Mozilla web browsers and in earlier simulation-based evaluation efforts.

Stefano D’Aronco <stefano.daronco@geod.baug.ethz.ch> of ETH Zurich (previously at Ecole Polytechnique Federale de Lausanne when contributing to this work) helped with implementation and evaluation of an early version of NADA in [ns-3].

Charles Ganzhorn <charles.ganzhorn@gmail.com> contributed to the testbed-based evaluation of NADA during an early stage of its development.

13. References

13.1. Normative References


13.2. Informative References

[Budzisz-TON11]

[Floyd-CCR00]

[I-D.ietf-avtcore-cc-feedback-message]

[I-D.ietf-rmcat-cc-codec-interactions]

[I-D.ietf-rmcat-cc-requirements]
Jesup, R. and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media", draft-ietf-rmcat-cc-requirements-09 (work in progress), December 2014.

[I-D.ietf-rmcat-cc-eval-test]

[I-D.ietf-rmcat-cc-wireless-tests]


Appendix A. Network Node Operations

NADA can work with different network queue management schemes and does not assume any specific network node operation. As an example, this appendix describes three variants of queue management behavior.
at the network node, leading to either implicit or explicit congestion signals. It needs to be acknowledged that NADA has not yet been tested with non-probabilistic ECN marking behaviors.

In all three flavors described below, the network queue operates with the simple first-in-first-out (FIFO) principle. There is no need to maintain per-flow state. The system can scale easily with a large number of video flows and at high link capacity.

A.1. Default behavior of drop tail queues

In a conventional network with drop tail or RED queues, congestion is inferred from the estimation of end-to-end delay and/or packet loss. Packet drops at the queue are detected at the receiver, and contributes to the calculation of the aggregated congestion signal \( x_{curr} \). No special action is required at network node.

A.2. RED-based ECN marking

In this mode, the network node randomly marks the ECN field in the IP packet header following the Random Early Detection (RED) algorithm [RFC7567]. Calculation of the marking probability involves the following steps:

\[
\begin{align*}
\text{on packet arrival:} \\
\quad & \text{update smoothed queue size } q_{\text{avg}} \text{ as:} \\
\quad & q_{\text{avg}} = w \cdot q + (1 - w) \cdot q_{\text{avg}}. \\
\quad & \text{calculate marking probability } p \text{ as:} \\
\quad & p = \begin{cases} \\
0, & \text{if } q < q_{\text{lo}}; \\
\frac{q_{\text{avg}} - q_{\text{lo}}}{q_{\text{hi}} - q_{\text{lo}}}, & \text{if } q_{\text{lo}} \leq q < q_{\text{hi}}; \\
1, & \text{if } q \geq q_{\text{hi}}. \\
\end{cases}
\end{align*}
\]

Here, \( q_{\text{lo}} \) and \( q_{\text{hi}} \) corresponds to the low and high thresholds of queue occupancy. The maximum marking probability is \( p_{\text{max}} \).

The ECN markings events will contribute to the calculation of an equivalent delay \( x_{curr} \) at the receiver. No changes are required at the sender.
A.3. Random Early Marking with Virtual Queues

Advanced network nodes may support random early marking based on a token bucket algorithm originally designed for Pre-Congestion Notification (PCN) [RFC6660]. The early congestion notification (ECN) bit in the IP header of packets are marked randomly. The marking probability is calculated based on a token-bucket algorithm originally designed for the Pre-Congestion Notification (PCN) [RFC6660]. The target link utilization is set as 90%; the marking probability is designed to grow linearly with the token bucket size when it varies between 1/3 and 2/3 of the full token bucket limit.

Calculation of the marking probability involves the following steps:

- upon packet arrival:
  - meter packet against token bucket \( (r, b) \);
  - update token level \( b_{tk} \);
  - calculate the marking probability as:

\[
\begin{align*}
    p &= \begin{cases}
        0, & \text{if } b - b_{tk} < b_{lo} \\
        \frac{b - b_{tk} - b_{lo}}{b_{hi} - b_{lo}}, & \text{if } b_{lo} \leq b - b_{tk} < b_{hi} \\
        1, & \text{if } b - b_{tk} \geq b_{hi}.
    \end{cases}
\end{align*}
\]

Here, the token bucket lower and upper limits are denoted by \( b_{lo} \) and \( b_{hi} \), respectively. The parameter \( b \) indicates the size of the token bucket. The parameter \( r \) is chosen to be below capacity, resulting in slight under-utilization of the link. The maximum marking probability is \( p_{\text{max}} \).

The ECN markings events will contribute to the calculation of an equivalent delay \( x_{\text{curr}} \) at the receiver. No changes are required at the sender. The virtual queuing mechanism from the PCN-based marking algorithm will lead to additional benefits such as zero standing queues.

Authors’ Addresses
Xiaoqing Zhu
Cisco Systems
12515 Research Blvd., Building 4
Austin, TX 78759
USA
Email: xiaoqzhu@cisco.com

Rong Pan *
* Pending affiliation change.
Email: rong.pan@gmail.com

Michael A. Ramalho
Cisco Systems, Inc.
8000 Hawkins Road
Sarasota, FL 34241
USA
Phone: +1 919 476 2038
Email: mar42@cornell.edu

Sergio Mena de la Cruz
Cisco Systems
EPFL, Quartier de l’Innovation, Batiment E
Ecublens, Vaud 1015
Switzerland
Email: semena@cisco.com
Abstract

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

Status of This Memo

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 https://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 April 29, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
   1.1. Wireless (LTE) access properties .......................... 3
   1.2. Why is it a self-clocked algorithm? ....................... 4
2. Terminology .................................................... 4
3. Overview of SCReAM Algorithm ................................. 4
   3.1. Network Congestion Control ............................... 7
   3.2. Sender Transmission Control .............................. 8
   3.3. Media Rate Control ...................................... 8
4. Detailed Description of SCReAM ............................... 9
   4.1. SCReAM Sender ........................................... 9
      4.1.1. Constants and Parameter values ...................... 9
      4.1.1.1. Constants ....................................... 10
      4.1.1.2. State variables .................................. 11
      4.1.2. Network congestion control ......................... 13
      4.1.2.1. Reaction to packets loss and ECN ................. 16
      4.1.2.2. Congestion window update ......................... 16
      4.1.2.3. Competing flows compensation ..................... 19
      4.1.2.4. Lost packet detection ............................ 21
      4.1.2.5. Send window calculation .......................... 22
      4.1.2.6. Packet pacing .................................. 23
      4.1.2.7. Resuming fast increase ............................ 23
      4.1.2.8. Stream prioritization ............................ 23
      4.1.3. Media rate control .................................. 24
4.2. SCReAM Receiver ........................................... 27
   4.2.1. Requirements on feedback elements ..................... 27
   4.2.2. Requirements on feedback intensity ..................... 29
5. Discussion ..................................................... 29
6. Implementation status ......................................... 30
   6.1. OpenWebRTC .............................................. 31
   6.2. A C++ Implementation of SCReAM .......................... 31
7. Suggested experiments ......................................... 32
8. Acknowledgements ............................................. 33
9. IANA Considerations .......................................... 33
10. Security Considerations ..................................... 33
11. Change history ............................................... 33
12. References ................................................... 34
   12.1. Normative References .................................... 35
   12.2. Informative References .................................. 35
Authors’ Addresses .............................................. 37
1. Introduction

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

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 can vary considerably for instance in cases where the wireless terminal is moving around. Even though LTE can support bitrates well above 100Mbps, there are cases when the available bitrate can be much lower, examples are situations with high network load and poor coverage. An additional complication is that the network throughput can drop for short time intervals at e.g. handover, these short glitches are initially very difficult to distinguish from more permanent reductions in throughput.

Unlike wireline bottlenecks with large statistical multiplexing it is not possible to try to maintain a given bitrate when congestion is
detected with the hope that other flows will yield, this is because there are generally few other flows competing for the same bottleneck. Each user gets its own variable throughput bottleneck, where the throughput depends on factors like channel quality, network load and historical throughput. The bottom line is, if the throughput drops, the sender has no other option than to reduce the bitrate. Once the radio scheduler has reduced the resource allocation for a bearer, an RMCAT flow in that bearer aims to reduce the sending rate quite quickly (within one RTT) in order to avoid excessive queuing delay or packet loss.

1.2. Why is it a self-clocked algorithm?

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

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

- A 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 still only one mechanism to adjust the sending rate. This makes it difficult to reach the goals of high throughput and prompt reaction to congestion.

2. 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].

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 number of bytes that can be transmitted at any given time instant. A congestion window puts an upper limit on how many bytes can be in flight, i.e. transmitted but not yet acknowledged.

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

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

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

It is sufficient that any of the two conditions above is fulfilled to make the base delay update properly. Furthermore [LEDBAT-delay-impact] describes an issue with short lived competing flows, the case in SCReAM is that these short lived flows will cause the self-clocking in SCReAM to slow down with the result that the RTP queue is built up, which will in turn result in a reduced media video bitrate. SCReAM will thus yield more to competing short lived flows than what is the case with traditional use of LEDBAT. The basic functionality in the use of LEDBAT in SCReAM is quite simple, there are however a few steps to take to make the concept work with conversational media:

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

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

- A qdelay 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 in the media rate calculation to achieve reduced jitter.

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

The above mentioned features will be described in more detail in sections Section 3.1 to Section 3.3. The full details are described in Section 4.
The SCReAM algorithm consists of three main parts: network congestion control, sender transmission control and media rate control. All of these three parts reside at the sender side. Figure 1 shows the functional overview of a SCReAM sender. The receiver side algorithm is very simple in comparison as it only generates feedback containing acknowledgements of received RTP packets and an ECN count.

3.1. Network Congestion Control

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

---

Figure 1: SCReAM sender functional view
The SCReAM congestion control method, uses techniques similar to LEDBAT [RFC6817] to measure the qdelay. As is the case with LEDBAT, it is not necessary to use synchronized clocks in sender and receiver in order to compute the qdelay. 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. Failure to meet this requirement leads to malfunction in the SCReAM congestion control algorithm due to incorrect estimation of the network queue delay.

The SCReAM sender calculates the congestion window based on the feedback from the SCReAM receiver. The congestion window is allowed to increase if the qdelay is below a predefined qdelay target, otherwise the congestion window decreases. The qdelay target is typically set to 50-100ms. 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. 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 estimated link throughput.

3.3. Media Rate Control

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

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

In cases where the sender frame queues increase rapidly such as in 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. Frame skipping results in the frame rate being temporarily reduced. Which method to use is a design choice 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 split between the network congestion control, sender transmission control and the media rate control.

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

Media frames are encoded and forwarded to the RTP queue (1) in Figure 1. The media rate adaptation adapts to the size of the RTP queue (2) and provides a target rate for the media encoder (3). The RTP packets are picked from the RTP queue (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) sends the RTP packets to the UDP socket (5). In the general case all media SHOULD go through the sender transmission controller and is limited so that the number of bytes in flight is less than the congestion window. RTCP packets are received (6) and the information about 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, within (), for the constants are deduced from experiments. The units are enclosed in square brackets [ ].

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

QDELAY_TARGET_HI (0.4s)
Target value for the maximum qdelay. This parameter provides an upper limit to how much the target qdelay (qdelay_target) can be increased in order to cope with competing loss based flows. The target qdelay does not have to be initialized to this high value however as it would increase e2e delay and also make the rate control and congestion control loop sluggish.

QDELAY_WEIGHT (0.1)
Averaging factor for qdelay_fraction_avg.

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

MIN_CWND (3000byte)
Minimum congestion window.

MAX_BYTES_IN_FLIGHT_HEAD_ROOM (1.1)
Headroom for the limitation of CWND.

GAIN (1.0)
Gain factor for congestion window adjustment.

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

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

BETA_R (0.9)
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], bps is bits per second.
TARGET_BITRATE_MAX
Max target bitrate [bps].

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

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

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

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

TARGET_RATE_SCALE_RTP_QDELAY (0.95) Target rate scale when RTP
qdelay threshold exceeds RTP_QDELAY_TH.

QDELAY_TREND_LO (0.2) Threshold value for qdelay_trend.

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

RATE_PACE_MIN (50000bps) Minimum pacing rate.

4.1.1.2. State variables

The values within () indicate initial values.

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 (Exponentially Weighted Moving Average) 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.

in_fast_increase (true)
True if in fast increase state.

cwnd (MIN_CWND)
Congestion window.

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

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

send_wnd (0)
Upper limit to how many bytes 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 with a similar method to that described in [RFC6298].

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

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

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

4.1.2. Network congestion control

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

- Computation of congestion window at the sender: Gives an upper limit to the number of bytes in flight.

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

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

The number of bytes in flight (bytes_in_flight) is computed as the sum of the sizes of the RTP packets ranging from the RTP packet most recently transmitted down to but not including the acknowledged packet with the highest sequence number. This can be translated to the difference between the highest transmitted byte sequence number and the highest acknowledged byte sequence number. As an example: If 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 variable is reset to zero after a CWND update.

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

- A list of received RTP packets’ sequence numbers.
- 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 a low pass filter function. A number of variables are updated as illustrated by the pseudo code below, temporary variables are appended with ‘_t’. As mentioned in Section 7, calculation of the proper congestion window and media bitrate may benefit from additional optimizations for handling of very high and very low bitrates, and from additional damping to handle periodic packet bursts. Some such optimizations are implemented in [SCReAM-CPP-implementation], but they do not form part of the specification of SCReAM at this time.
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_t’ has positive values if qdelay shows an increasing or decreasing trend, thus an indication of congestion is obtained before the qdelay target is reached. As a side effect, also the case that qdelay decreases is taken as a sign of congestion, experiments have however shown that this is beneficial as varying queue delay up or down is an indication that the transmit rate is very close to the path capacity.

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

\[ R(x,k) = \sum_{n=1}^{n=N-k} x(n) \cdot x(n+k) \]

The prediction coefficient is further multiplied with qdelay_fraction_avg to reduce sensitivity to increasing qdelay when it is very small. The 50ms sampling is a simplification that could have the effect that the same qdelay is sampled several times, this does however not pose any problem as the vector is only used to determine if the qdelay is increasing or decreasing. The
qdelay_trend is utilized in the media rate control to indicate incipient congestion and to determine when to exit from fast increase mode. qdelay_trend_mem is used to enforce a less aggressive rate increase after congestion events. The function update_qdelay_fraction_hist(..) removes the oldest element and adds the latest qdelay_fraction element to the qdelay_fraction_hist vector.

4.1.2.1. Reaction to packets loss and ECN

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

The congestion window back off due to loss events is deliberately a bit less than is the case with e.g. TCP Reno. 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 whose rate is limited to a value close to the available transmit rate and often below that 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 RECOMMENDED to let SCReAM reduce the congestion window less than what is the case with TCP when loss events occur.

An ECN event is detected if the n_ECN counter in the feedback report has increased since the previous received feedback. Once an ECN event is detected, the n_ECN counter is ignored for a full smoothed round trip time, the intention of this is to limit the congestion window decrease to at most once per round trip. The congestion window back off due to an ECN event MAY be smaller than if a loss event occurs. This is in line with the idea outlined in [I-D.ietf-tcpm-alternativebackoff-ecn] to enable ECN marking thresholds lower than the corresponding packet drop thresholds.

4.1.2.2. Congestion window update

The update of the congestion window depends on whether loss or ECN-marking or neither occurs. The pseudo code below describes actions taken in case of the different events.
<CODE BEGINS>
    on congestion event(qdelay):
        # Either loss or ECN mark is detected
        in_fast_increase = false
        if (is loss)
            # Loss is detected
            cwnd = max(MIN_CWND, cwnd*BETA_LOSS)
        else
            # No loss, so it is then an ECN mark
            cwnd = max(MIN_CWND, cwnd*BETA_ECN)
        end
        adjust_qdelay_target(qdelay) #compensating for competing flows
        calculate_send_window(qdelay, qdelay_target)

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

The methods are further described in detail below.

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

Pseudo code for the update of the congestion window is found below.
update_cwnd(bytes_newly_acked):

    # In fast increase?
    if (in_fast_increase)
        if (qdelay_trend >= QDELAY_TREND_TH)
            # Incipient congestion detected, exit fast increase
            in_fast_increase = false
        else
            # No congestion yet, increase cwnd if it
            # is sufficiently used
            # An additional slack of bytes_newly_acked is
            # added to ensure that CWND growth occurs
            # even when feedback is sparse
            if (bytes_in_flight*1.5+bytes_newly_acked > cwnd)
                cwnd = cwnd+bytes_newly_acked
            end
            return
        end
    end

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

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

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

CWND is updated differently depending on whether the congestion control is in fast increase state or not, as controlled by the variable in_fast_increase.
When in fast increase state, the congestion window is increased with the number of newly acknowledged bytes as long as the window is sufficiently used. Sparse feedback can potentially limit congestion window growth, an additional slack is therefore added, given by the number of newly acknowledged bytes.

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

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

Further the CWND is limited by max_bytes_in_flight and MIN_CWND. The limitation of the congestion window by the maximum number of bytes in flight over the last 5 seconds (max_bytes_in_flight) avoids possible 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.

### 4.1.2.3. 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, examples are large FTP flows using loss based congestion control. The worst condition occurs when the bottleneck queues are of tail-drop type with a large buffer size. SCReAM takes care of such situations by adjusting the qdelay_target when loss based flows are detected, as given by the pseudo code below.
Two temporary variables are calculated. qdelay_norm_avg_t is the long term average queue delay, qdelay_norm_var_t is the long term variance of the queue delay. A high qdelay_norm_var_t indicates that the queue delay changes, this can be an indication of reduced bottleneck bandwidth or that a competing flow has just entered. Thus, it indicates that it is not safe to adjust the queue delay target.

A low qdelay_norm_var_t indicates that the queue delay is relatively stable, the reason can be that the queue delay is low but it can also be an indication that a competing flow is filling up the bottleneck.
to the limit where packet losses may start to occur, in which case the queue delay will stay relatively high for a longer time.

The queue delay target is allowed to be increased if, either the loss event rate is above a given threshold or that qdelay_norm_var_t is low. Both these conditions indicate that a competing flow may be present. In all other cases the queue delay target is decreased.

The function that adjusts the qdelay_target is simple and has a certain risk to produce both false positive and negatives. The case that self-inflicted congestion by the SCReAM algorithm may be falsely interpreted as the presence of competing loss based FTP flows is a false positive. The opposite case where the algorithm fails to detect the presence of a competing FTP flow is a false negative.

Extensive simulations have shown that the algorithm performs well in LTE test cases and that it also performs well in simple bandwidth limited bottleneck test cases with competing FTP flows. It can however not be completely ruled out that this algorithm can fail. Especially the false positives can be problematic as the end to end delay can increase dramatically if the target queue delay is increased by accident as a result of self-inflicted congestion.

If it is deemed unlikely that competing flows occur over the same bottleneck, the algorithm described in this section MAY be turned off. One such case can be QoS enabled bearers in 3GPP based access such as LTE. However, when sending over the Internet, often the network conditions are not known for sure and it is in general not possible to make safe assumptions on how a network is used and whether or not competing flows share the same bottleneck. Therefore turning this algorithm off must be considered with caution as that can lead to basically zero throughput if competing with other, loss based, traffic.

4.1.2.4. Lost packet detection

Lost packet detection is based on the received sequence number list. A reordering window SHOULD be applied to avoid that packet reordering triggers loss events.

The reordering window is specified as a time unit, similar to the ideas behind RACK (Recent ACKnowledgement) [I-D.ietf-tcpm-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 acknowledged.

4.1.2.5. Send window calculation

The basic design principle behind packet transmission in SCReAM is to allow transmission only if the number of bytes in flight is less than the congestion window. There are however two reasons why this strict rule will not work optimally:

- Bitrate variations: Media sources such as video encoders generally produce frames whose size always vary to a larger or smaller extent. The RTP queue absorbs the natural variations in frame sizes. The RTP queue should however be as short as possible, to avoid that the end to end delay increases. To achieve that, the media rate control takes the RTP queue size into account when the target bitrate for the media is computed. A strict ‘send only when bytes in flight is less than the congestion window’ rule can lead to that the RTP queue grows simply because the send window is limited, the effect of which would be that the target bitrate is pushed down. The consequence of this is that the congestion window will not increase, or will increase very slowly, because the congestion window is only allowed to increase when there is a sufficient amount of data in flight. The end effect is then that the media bitrate increases very slowly or not at all.

- Reverse (feedback) path congestion: Especially in transport over buffer-bloated networks, the one way delay in the reverse direction can jump due to congestion. The effect 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 congestion window and the number of bytes in flight. A strict rule is applied when qdelay is higher than qdelay_target, to avoid further queue buildup in the network. For cases when qdelay is lower than the qdelay_target, a more relaxed rule is applied. This allows the bitrate to increase quickly when no congestion is detected while still being able to 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
    end

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

4.1.2.6.  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 also mitigates possible issues with queue overflow due to key-frame generation in video coders. The time interval between consecutive packet transmissions is enforced to be equal to or higher than \( t_{pace} \) where \( t_{pace} \) is given by the equations below:

\[
pace_{bitrate} = \max (RATE_{PACE\_MIN}, \frac{cwnd \times 8}{s_{rtt}}) \\
t_{pace} = \frac{rtp_{size} \times 8}{pace_{bitrate}}
\]

\( rtp_{size} \) is the size of the last transmitted RTP packet, \( s_{rtt} \) is the smoothed round trip time. \( RATE_{PACE\_MIN} \) is the minimum pacing rate.

4.1.2.7.  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.

4.1.2.8.  Stream prioritization

The SCReAM algorithm makes a good distinction between network congestion control and the media rate control. 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...
The implementation of SCReAM [SCReAM-CPP-implementation] use credit based scheduling. In credit based scheduling, credit is accumulated by queues as they wait for service and are spent while the queues are being serviced. For instance, if one queue is allowed to transmit 1000 bytes, then a credit of 1000 bytes is allocated to the other unscheduled queues. This principle can be extended to weighted scheduling in which case the credit allocated to unscheduled queues depends on the relative weights. The latter is also implemented in [SCReAM-CPP-implementation].

4.1.3. Media rate control

The media rate control algorithm is executed at regular intervals 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 to reduce the amount of induced network jitter.

The role of the media rate control is to strike a reasonable balance between a low amount of queuing in the RTP queue(s) and a sufficient amount of data to send in order to keep the data path busy. A too cautious setting leads to possible under-utilization of network capacity leading to that the flow can become starved out by other more opportunistic traffic. On the other hand, a too aggressive setting leads to increased jitter.

The target bitrate is adjusted depending on the congestion state. The target bitrate can vary between a minimum value (TARGET_BITRATE_MIN) and a maximum value (TARGET_BITRATE_MAX). TARGET_BITRATE_MIN SHOULD be chosen to a low enough value to avoid that RTP packets become queued up when the network throughput is reduced. The sender SHOULD also be equipped with a mechanism that discards RTP packets in cases where 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 second.

Both estimates are updated every 200 ms.
The current throughput, current_rate, is computed as the maximum value of rate_transmit and rate_ack. The rationale behind the use of rate_ack in addition to rate_transmit is that rate_transmit is affected also by the amount of data that is available to transmit, thus a lack of data to transmit can be seen as reduced throughput that can 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 or ECN event has occurred and if the congestion control is in fast increase or not.

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

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

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

ramp_up_speed_t = min(RAMP_UP_SPEED, target_bitrate/2.0)
scale_t = (target_bitrate - target_bitrate_last_max)/
          target_bitrate_last_max
scale_t = max(0.2, min(1.0, (scale_t*4)^2))
# min scale_t value 0.2 as the bitrate should be allowed to
# increase at least slowly --> avoid locking the rate to
# target_bitrate_last_max
if (in_fast_increase = true)
  increment_t = ramp_up_speed_t*RATE_ADJUST_INTERVAL
  increment_t *= scale_t
  target_bitrate += increment_t
else
  current_rate_t = max(rate_transmit, rate_ack)
  # Compute a bitrate change
  delta_rate_t = current_rate_t*(1.0-PRE_CONGESTION_GUARD*
                               queue_delay_trend)-TX_QUEUE_SIZE_FACTOR *rtp_queue_size
  # Limit a positive increase if close to target_bitrate_last_max
  if (delta_rate_t > 0)
    delta_rate_t *= scale_t
    delta_rate_t =
                   min(delta_rate_t, ramp_up_speed_t*RATE_ADJUST_INTERVAL)
  end
  target_bitrate += delta_rate_t
</CODE>
# Force a slight reduction in bitrate if RTP queue builds up

```c
rtp_queue_delay_t = rtp_queue_size/current_rate_t
if (rtp_queue_delay_t > RTP_QDELAY_TH)
    target_bitrate *= TARGET_RATE_SCALE_RTP_QDELAY
end
```

```c
rate_media_limit_t =
    max(current_rate_t, max(rate_media, rtp_rate_median))
rate_media_limit_t *= (2.0-qdelay_trend_mem)
target_bitrate = min(target_bitrate, rate_media_limit_t)
target_bitrate = min(TARGET_BITRATE_MAX,  
    max(TARGET_BITRATE_MIN,target_bitrate))
```

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 RTP packets being queued up when the transmit rate decreases due to the reduced congestion window. A similar rate reduction happens when ECN events are detected.

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

When in fast increase state (in_fast_increase=true), the bitrate increase is given by the desired ramp-up speed (RAMP_UP_SPEED). The ramp-up speed is limited when the target bitrate is low to avoid rate oscillation at low bottleneck bitrates. The setting of RAMP_UP_SPEED depends on preferences, a high setting such as 1000kbps/s makes it possible to quickly get high quality media, this is however at the expense of a increased 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 when congestion is detected.
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. As these parameters are used also in the network congestion control one may suspect some odd interaction between the media rate control and the network congestion control, this is in fact the case if the parameter PRE_CONGESTION_GUARD is set to a high value. The use of qdelay_trend in the media rate control is solely to reduce jitter, the dependency can be removed by setting PRE_CONGESTION_GUARD=0, the effect is a somewhat faster rate increase after congestion, at the expense of increased jitter in congested situations.

4.2. SCReAM Receiver

The simple task of the SCReAM receiver is to feedback acknowledgements of received packets and total ECN count to the SCReAM sender, in addition, the receive time of the RTP packet with the highest sequence number is echoed back. Upon reception of each RTP packet the receiver MUST maintain enough information to send the aforementioned values to the SCReAM sender via a RTCP transport layer feedback message. The frequency of the feedback message depends on the available RTCP bandwidth. The requirements on the feedback elements and the feedback interval is described.

4.2.1. Requirements on feedback elements

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

- A list of received RTP packets. This list SHOULD be sufficiently long to cover all received RTP packets. This list can be realized with the Loss RLE 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 qdelay. This can be realized by means of the Packet Receipt Times Report Block in [RFC3611]. begin_seq MUST be set to the highest received (possibly wrapped around) sequence number, end_seq MUST be set to begin_seq+1 % 65536. The timestamp clock MAY be set according to [RFC3611] 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.
In a typical use case, no more than four Loss RLE chunks are needed, thus the feedback message will be 44 bytes. It is obvious from 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, could 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 a counter that increments by one for each received packet with the ECN-CE code point set. The more bulky format could nevertheless be useful for e.g. ECN black-hole detection.
4.2.2. Requirements on feedback intensity

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

The feedback interval depends on the media bitrate. At low bitrates it is sufficient with a feedback interval of 100 to 400ms, while at high bitrates a feedback interval of roughly 20ms is to prefer, at very high bitrates, even shorter feedback intervals MAY be needed in order to keep the self-clocking in SCReAM working well. One piece of evidence of a too sparse feedback is that the SCReAM implementation cannot reach high bitrates, even in uncongested links. A more frequent feedback might solve this issue.

The numbers above can be formulated as feedback interval function that can be useful for the computation of the desired RTCP bandwidth. The following equation expresses the feedback rate:

\[
rate_{fb} = \min(50, \max(2.5, rate_{media}/10000))
\]

rate_media is the RTP media bitrate expressed in \([\text{bits/s}]\), rate_fb is the feedback rate expressed in \([\text{packets/s}]\). Converted to feedback interval we get:

\[
fb_{int} = 1.0/\min(50, \max(2.5, rate_{media}/10000))
\]

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.

SCReAM works with AVPF regular mode, immediate or early mode is not required by SCReAM but can nonetheless be useful for e.g RTCP messages not directly related to SCReAM, such as those specified in [RFC4585]. It is RECOMMENDED to use reduced size RTCP [RFC5506] where regular full compound RTCP transmission is controlled by trr-int as described in [RFC4585].

5. Discussion

This section covers a few discussion points
Clock drift: SCReAM can suffer from the same issues with clock drift as is the case with LEDBAT [RFC6817]. Section A.2 in [RFC6817] however describes ways to mitigate issues with clock drift.

Support for alternate ECN semantics: This specification adopts the proposal in [I-D.ietf-tcpm-alternativebackoff-ecn] to reduce the congestion window less when ECN based congestion events are detected. Future work on Low Loss Low Latency for Scalable throughput (L4S) may lead to updates in a future RFC that describes SCReAM support for L4S.

A new RFC4585 transport layer feedback message could to be standardized if the use of the already existing RTCP extensions as described in Section 4.2 is not deemed sufficient.

The target bitrate given by SCReAM depicts the bitrate including RTP and FEC overhead. The media encoder SHOULD take this overhead into account when the media bitrate is set. This means that the media coder bitrate SHOULD be computed as

\[
\text{media\_rate} = \text{target\_bitrate} - \text{rtp\_plus\_fec\_overhead\_bitrate}
\]

It is not strictly necessary to make a 100% perfect compensation for the overhead as the SCReAM algorithm will inherently compensate for moderate errors. Under-compensation of the overhead has the effect of increasing jitter while overcompensation will have the effect of causing the bottleneck link to become under-utilized.

6. Implementation status

[Editor’s note: Please remove the whole section before publication, as well reference to RFC 7942]

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 [RFC7942]. 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 [RFC7942], "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 the specification in this memo. 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 available at[SCReAM-CPP-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 that allows for some initial experiments. An additional experiment with SCReAM in a remote control arrangement is also documented.

- **Coverage**: The code implements the specification in this memo.
- **Contact**: ingemar.s.johansson@ericsson.com

### 7. Suggested experiments

SCReAM has been evaluated in a number of different ways, most of the evaluation has been in simulator. The OpenWebRTC implementation work involved extensive testing with artificial bottlenecks with varying bandwidths and using two different video coders (OpenH264 and VP9), the experience of this lead to further improvements of the media rate control logic.

Further experiments are preferably done by means of implementation in real clients and web browsers. RECOMMENDED experiments are:

- **Trials with various access technologies**: EDGE/3G/4G, WiFi, DSL. Some experiments have already been carried out with LTE access, see e.g. [SCReAM-CPP-implementation] and [SCReAM-implementation-experience]
- **Trials with different kinds of media**: Audio, Video, slide show content. Evaluation of multi stream handling in SCReAM.
- **Evaluation of functionality of competing flows compensation mechanism**: Evaluate how SCReAM performs with competing TCP like traffic and to what extent the competing flows compensation causes self-inflicted congestion.
- **Determine proper parameters**: A set of default parameters are given that makes SCReAM work over a reasonably large operation range, however for instance for very low or very high bitrates it may be necessary to use different values for instance for the RAMP_UP_SPEED.
- **Experimentation with further improvements to the congestion window and media bitrate calculation**: [SCReAM-CPP-implementation] implements some optimizations, not described in this memo, that improve performance slightly. Further experiments are likely to lead to more optimizations of the algorithm.
8. 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 RMCAT chairs Karen E. Nielsen and Mirja Kuehlewind for patiently reading, suggesting improvements and also for asking all the difficult but necessary questions. Thanks to Stefan Holmer, Xiaoqing Zhu, Saifqul Islam and David Hayes for the additional review of this document. Thanks to Ralf Globisch for taking time to try out SCReAM in his challenging low bitrate use cases, Robert Hedman for finding a few additional flaws in the running code, and Gustavo Garcia and ‘miseri’ for code contributions.

9. IANA Considerations

There is currently no request to IANA

10. 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. This attack is however mitigated by the minimum send rate maintained by SCReAM when no feedback is received.

11. Change history

A list of changes:

o WG-12 to WG-13: IESG comments addressed

o WG-11 to WG-12: Review comments from Joel Halpern and Mirja

o WG-10 to WG-11: Review comments from Mirja

o WG-9 to WG-10: Minor edits

o WG-08 to WG-09: Updated based shepherd review by Martin Stiemerling, Q-bit semantics are removed as this is superfluous for the moment. Pacing and RTCP considerations are moved up from the appendix, FEC discussion moved to discussion section.
o WG-07 to WG-08: Avoid draft expiry

o WG-06 to WG-07: Updated based on WGLC review by David Hayes and Safiqul Islam

o WG-05 to WG-06: Added list of suggested experiments

o WG-04 to WG-05: Congestion control and rate control simplified somewhat

o WG-03 to WG-04: Editorial fixes

o 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.

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

o 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

12. References
12.1. Normative References


12.2. Informative References


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


Johansson & Sarker       Expires April 29, 2018    [Page 36]

[QoS-3GPP]
[RFC6679]
[RFC7478]


Authors' Addresses

Ingemar Johansson
Ericsson AB
Laboratoriegården 11
Luleåa 977 53
Sweden

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