RTP Media Congestion Avoidance Techniques — D. Hayes, Ed., University of Oslo
Internet-Draft — S. Ferlin, Simula Research Laboratory
Intended status: Experimental — M. Welzl, University of Oslo
Expires: September 4, 2015 — March 3, 2015

Shared Bottleneck Detection for Coupled Congestion Control for RTP Media.
draft-hayes-rmcat-sbd-02

Abstract

This document describes a mechanism to detect whether end-to-end data flows share a common bottleneck. It relies on summary statistics that are calculated by a data receiver based on continuous measurements and regularly fed to a grouping algorithm that runs wherever the knowledge is needed. This mechanism complements the coupled congestion control mechanism in draft-welzl-rmcat-coupled-cc.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In the Internet, it is not normally known if flows (e.g., TCP connections or UDP data streams) traverse the same bottlenecks. Even flows that have the same sender and receiver may take different paths and share a bottleneck or not. Flows that share a bottleneck link usually compete with one another for their share of the capacity. This competition has the potential to increase packet loss and delays. This is especially relevant for interactive applications that communicate simultaneously with multiple peers (such as multi-party video). For RTP media applications such as RTCWEB, [I-D.welzl-rmcat-coupled-cc] describes a scheme that combines the congestion controllers of flows in order to honor their priorities and avoid unnecessary packet loss as well as delay. This mechanism relies on some form of Shared Bottleneck Detection (SBD); here, a measurement-based SBD approach is described.

1.1. The signals

The current Internet is unable to explicitly inform endpoints as to which flows share bottlenecks, so endpoints need to infer this from whatever information is available to them. The mechanism described here currently utilises packet loss and packet delay, but is not restricted to these.

1.1.1. Packet Loss

Packet loss is often a relatively rare signal. Therefore, on its own it is of limited use for SBD, however, it is a valuable supplementary measure when it is more prevalent.

1.1.2. Packet Delay

End-to-end delay measurements include noise from every device along the path in addition to the delay perturbation at the bottleneck device. The noise is often significantly increased if the round-trip time is used. The cleanest signal is obtained by using One-Way-Delay (OWD).

Measuring absolute OWD is difficult since it requires both the sender and receiver clocks to be synchronised. However, since the statistics being collected are relative to the mean OWD, a relative OWD measurement is sufficient. Clock drift is not usually significant over the time intervals used by this SBD mechanism (see [RFC6817] A.2 for a discussion on clock drift and OWD measurements). However, in circumstances where it is significant, Section 3.3.2 outlines a way of adjusting the calculations to cater for it.

Each packet arriving at the bottleneck buffer may experience very different queue lengths, and therefore different waiting times. A single OWD sample does not, therefore, characterize the path well. However, multiple OWD measurements do reflect the distribution of delays experienced at the bottleneck.

1.1.3. Path Lag

Flows that share a common bottleneck may traverse different paths, and these paths will often have different base delays. This makes it difficult to correlate changes in delay or loss. This technique uses the long term shape of the delay distribution as a base for comparison to counter this.

2. Definitions

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 RFC 2119 [RFC2119].

Acronyms used in this document:

OWD -- One Way Delay

PDV -- Packet Delay Variation

RTT -- Round Trip Time

SBD -- Shared Bottleneck Detection

Conventions used in this document:

T -- the base time interval over which measurements are made.

N -- the number of base time, T, intervals used in some calculations.

sum_T(...) -- summation of all the measurements of the variable in parentheses taken over the interval T

sum(...) -- summation of terms of the variable in parentheses

sum_N(...) -- summation of N terms of the variable in parentheses sum_NT(...) -- summation of all measurements taken over the interval N*T

E_T(...) -- the expectation or mean of the measurements of the variable in parentheses over T

E_N(...) -- The expectation or mean of the last N values of the variable in parentheses

E_M(...) -- The expectation or mean of the last M values of the variable in parentheses, where M <= N.

max_T(...) -- the maximum recorded measurement of the variable in parentheses taken over the interval T

min_T(...) -- the minimum recorded measurement of the variable in parentheses taken over the interval T

num_T(...) -- the count of measurements of the variable in parentheses taken in the interval T

num_VM(...) -- the count of valid values of the variable in parentheses given M records

PC -- a boolean variable indicating the particular flow was identified as experiencing congestion in the previous interval T (i.e. Previously Congested)

CD_T -- an estimate of the effect of Clock Drift on the mean OWD per T

CD_Adj(...) -- Mean OWD adjusted for clock drift

p_l, p_f, p_pdv, c_s, c_h, p_s, p_d, p_v -- various thresholds used in the mechanism.

N, M, and F -- number of values (calculated over T).

2.1. Parameter Values

Reference [Hayes-LCN14] uses T=350ms, N=50, p_l = 0.1. The other parameters have been tightened to reflect minor enhancements to the algorithm outlined in Section 3.3: c_s = -0.01, p_f = p_s = p_d = 0.1, p_pdv = 0.2, p_v = 0.2. M=50, F=10, and c_h = 0.3 are additional parameters defined in the document. These are values that seem to work well over a wide range of practical Internet conditions, but are the subject of ongoing tests.

3. Mechanism

The mechanism described in this document is based on the observation that the distribution of delay measurements of packets from flows that share a common bottleneck have similar shape characteristics. These shape characteristics are described using 3 key summary statistics:

variance (estimate var_est, see Section 3.1.3)

skewness (estimate skew_est, see Section 3.1.2)

oscillation (estimate freq_est, see Section 3.1.4)

with packet loss (estimate pkt_loss, see Section 3.1.5) used as a supplementary statistic.

Summary statistics help to address both the noise and the path lag problems by describing the general shape over a relatively long period of time. This is sufficient for their application in coupled congestion control for RTP Media. They can be signalled from a receiver, which measures the OWD and calculates the summary statistics, to a sender, which is the entity that is transmitting the media stream. An RTP Media device may be both a sender and a receiver. SBD can be performed at either Sender or receiver or both.

```
                  +----+
                  | H2 |
                  +----+
                     |
                     | L2
                     |
         +----+  L1  |  L3  +----+
         | H1 |------|------| H3 |
         +----+             +----+
```

A network with 3 hosts (H1, H2, H3) and 3 links (L1, L2, L3).

Figure 1

In Figure 1, there are two possible cases for shared bottleneck detection: a sender-based and a receiver-based case.

1. Sender-based: consider a situation where host H1 sends media streams to hosts H2 and H3, and L1 is a shared bottleneck. H2 and H3 measure the OWD and calculate summary statistics, which they send to H1 every T. H1, having this knowledge, can determine the shared bottleneck and accordingly control the send rates.

2. Receiver-based: consider that H2 is also sending media to H3, and L3 is a shared bottleneck. If H3 sends summary statistics to H1 and H2, neither H1 nor H2 alone obtain enough knowledge to detect this shared bottleneck; H3 can however determine it by combining the summary statistics related to H1 and H2, respectively. This case is applicable when send rates are controlled by the receiver; then, the signal from H3 to the senders contains the sending rate.

A discussion of the required signalling for the receiver-based case is beyond the scope of this document. For the sender-based case, the messages and their data format will be defined here in future versions of this document. We envision that an initialization message from the sender to the receiver could specify which key metrics are requested out of a possibly extensible set (pkt_loss, var_est, skew_est, freq_est). The grouping algorithm described in this document requires all four of these metrics, and receivers MUST be able to provide them, but future algorithms may be able to exploit other metrics (e.g. metrics based on explicit network signals). Moreover, the initialization message could specify T, N, and the necessary resolution and precision (number of bits per field).

3.1. Key metrics and their calculation

Measurements are calculated over a base interval, T. T should be long enough to provide enough samples for a good estimate of skewness, but short enough so that a measure of the oscillation can be made from N of these estimates. Reference [Hayes-LCN14] uses T = 350ms and N=M=50, which are values that seem to work well over a wide range of practical Internet conditions.

3.1.1. Mean delay

The mean delay is not a useful signal for comparisons between flows since flows may traverse quite different paths and clocks will not necessarily be synchronized. However, it is a base measure for the 3 summary statistics. The mean delay, E_T(OWD), is the average one way delay measured over T.

To facilitate the other calculations, the last N E_T(OWD) values will need to be stored in a cyclic buffer along with the moving average of E_T(OWD):

    mean_delay = E_M(E_T(OWD)) = sum_M(E_T(OWD)) / M

where M <= N. Generally M=N, setting M to be less than N allows the mechanism to be more responsive to changes, but potentially at the expense of a higher error rate (see Section 3.4 for a discussion on improving the responsiveness of the mechanism.)

3.1.2. Skewness Estimate

Skewness is difficult to calculate efficiently and accurately. Ideally it should be calculated over the entire period (M * T) from the mean OWD over that period. However this would require storing every delay measurement over the period. Instead, an estimate is made over T using the previous calculation of mean_delay. Comparisons are made using the mean of M skew estimates (an alternative that removes bias in the mean is given in Section 3.3.3).

The skewness is estimated using two counters, counting the number of one way delay samples (OWD) above and below the mean:

```
   skew_est_T =  (sum_T(OWD < mean_delay)

                 - sum_T(OWD > mean_delay)) / num_T(OWD)

      where

         if (OWD < mean_delay) 1 else 0

         if (OWD > mean_delay) 1 else 0

      skew_est_T is a number between -1 and 1

   skew_est = E_M(skew_est_T) = sum_M(skew_est_T) / M
```

For implementation ease, mean_delay does not include the mean of the current T interval.

Note: Care must be taken when implementing the comparisons to ensure that rounding does not bias skew_est. It is important that the mean is calculated with a higher precision than the samples.

3.1.3. Variance Estimate

Packet Delay Variation (PDV) ([RFC5481] and [ITU-Y1540]) is used as an estimator of the variance of the delay signal. We define PDV as follows:

PDV = PDV_max = max_T(OWD) - E_T(OWD)

var_est = E_M(PDV) = sum_M(PDV) / M

This modifies PDV as outlined in [RFC5481] to provide a summary statistic version that best aids the grouping decisions of the algorithm (see [Hayes-LCN14] section IVB).

The use of PDV = PDV_min = E_T(OWD) - min_T(OWD) is currently being investigated as an alternative that is less sensitive to noise. The drawback of using PDV_min is that it does not distinguish between groups of flows with similar values of skew_est as well as PDV_max (see [Hayes-LCN14] section IVB).

3.1.4. Oscillation Estimate

An estimate of the low frequency oscillation of the delay signal is calculated by counting and normalising the significant mean, E_T(OWD), crossings of mean_delay:

freq_est = number_of_crossings / N

Where

we define a significant mean crossing as a crossing that extends p_v * var_est from mean_delay. In our experiments we have found that p_v = 0.2 is a good value.

Freq_est is a number between 0 and 1. Freq_est can be approximated incrementally as follows:

With each new calculation of E_T(OWD) a decision is made as to whether this value of E_T(OWD) significantly crosses the current long term mean, mean_delay, with respect to the previous significant mean crossing.

A cyclic buffer, last_N_crossings, records a 1 if there is a significant mean crossing, otherwise a 0.

The counter, number_of_crossings, is incremented when there is a significant mean crossing and subtracted from when a non-zero value is removed from the last_N_crossings.

This approximation of freq_est was not used in [Hayes-LCN14], which calculated freq_est every T using the current E_N(E_T(OWD)). Our tests show that this approximation of freq_est yields results that are almost identical to when the full calculation is performed every T.

3.1.5. Packet loss

The proportion of packets lost is used as a supplementary measure:

pkt_loss = sum_NT(lost packets) / sum_NT(total packets)

Note: When pkt_loss is small it is very variable, however, when pkt_loss is high it becomes a stable measure for making grouping decisions.

3.2. Flow Grouping

3.2.1. Flow Grouping Algorithm

The following grouping algorithm is RECOMMENDED for SBD in the RMCAT context and is sufficient and efficient for small to moderate numbers of flows. For very large numbers of flows (e.g. hundreds), a more complex clustering algorithm may be substituted.

Since no single metric is precise enough to group flows (due to noise), the algorithm uses multiple metrics. Each metric offers a different "view" of the bottleneck link characteristics, and used together they enable a more precise grouping of flows than would otherwise be possible.

Flows determined to be experiencing congestion are successively divided into groups based on freq_est, var_est, and skew_est.

The first step is to determine which flows are experiencing congestion. This is important, since if a flow is not experiencing congestion its delay based metrics will not describe the bottleneck, but the "noise" from the rest of the path. Skewness, with proportion of packets loss as a supplementary measure, is used to do this:

1. Grouping will be performed on flows where:

```
skew_est < c_s

   || ( skew_est < c_h && PC )

   || pkt_loss > p_l
```

The parameter c_s controls how sensitive the mechanism is in detecting congestion. C_s = 0.0 was used in [Hayes-LCN14]. A value of c_s = 0.05 is a little more sensitive, and c_s = -0.05 is a little less sensitive. C_h controls the hysteresis on flows that were grouped as experiencing congestion last time.

These flows, flows experiencing congestion, are then progressively divided into groups based on the freq_est, PDV, and skew_est summary statistics. The process proceeds according to the following steps:

2. Group flows whose difference in sorted freq_est is less than a threshold:

```
diff(freq_est) < p_f
```

3. Group flows whose difference in sorted E_N(PDV) (highest to lowest) is less than a threshold:

```
diff(var_est) < (p_pdv * var_est)
```

   The threshold, (p_pdv * var_est), is with respect to the highest value in the difference.

4. Group flows whose difference in sorted skew_est or pkt_loss is less than a threshold:

```
if pkt_loss < p_l

   diff(skew_est) < p_s

otherwise

   diff(pkt_loss) < (p_d * pkt_loss)
```

   The threshold, (p_d * pkt_loss), is with respect to the highest value in the difference.

This procedure involves sorting estimates from highest to lowest. It is simple to implement, and efficient for small numbers of flows, such as are expected in RTCWEB.

3.2.2. Using the flow group signal

A grouping decisions is made every T from the second T, though they will not attain their full design accuracy until after the N'th T interval.

Network conditions, and even the congestion controllers, can cause bottlenecks to fluctuate. A coupled congestion controller MAY decide only to couple groups that remain stable, say grouped together 90% of the time, depending on its objectives. Recommendations concerning this are beyond the scope of this draft and will be specific to the coupled congestion controllers objectives.

3.3. Removing Noise from the Estimates

The following describe small changes to the calculation of the key metrics that help remove noise from them. Currently these "tweaks" are described separately to keep the main description succinct. In future revisions of the draft these enhancements may replace the original key metric calculations.

3.3.1. Oscillation noise

When a path has no congestion, the PDV will be very small and the recorded significant mean crossings will be the result of path noise. Thus up to N-1 meaningless mean crossings can be a source of error at the point a link becomes a bottleneck and flows traversing it begin to be grouped.

To remove this source of noise from freq_est:

1. Set the current PDV to PDV = NaN (a value representing an invalid record, ie Not a Number) for flows that are deemed to not be experiencing congestion by the first skew_est based grouping test (see Section 3.2.1).

2. Then var_est = sum_M(PDV != NaN) / num_VM(PDV)

3. For freq_est, only record a significant mean crossing if flow is experiencing congestion.

These three changes will remove the non-congestion noise from freq_est.

3.3.2. Clock drift

Generally sender and receiver clock drift will be too small to cause significant errors in the estimators. Skew_est is most sensitive to this type of noise. In circumstances where clock drift is high, making M < N can reduce this error.

A better method is to estimate the effect the clock drift is having on the E_N(E_T(OWD)), and then adjust mean_delay accordingly. A simple method of doing this follows:

First divide the N E_T(OWD) values into two halves (N/2 in each) -- old and new.

Calculate a mean of the old half:

Older_mean = E_old(E_T(OWD)) / N/2

Calculate a mean of the new (most recent) half:

Newer_mean = E_new(E_T(OWD)) / N/2

A linear estimate of the Clock Drift per T estimates is:

CD_T = (Newer_mean - Older_mean)/N/2

An adjusted mean estimate then is:

mean_delay = CD_Adj(E_M(E_T(OWD))) = E_M(E_T(OWD)) + CD_T * M/2

CD_Adj can be thought of as a prediction of what the long term mean will be in the current measurement period T. It is used as the basis for skew_est and freq_est.

3.3.3. Bias in the skewness measure

If successive calculations of skew_est are made with very different numbers of samples (num_T(OWD)), the simple calculation of E_M(skew_est) used for grouping decisions will be biased by the intervals that have few samples samples. This bias can be corrected if necessary as follows.

```
    skew_base_T = (sum_T(OWD < mean_delay

                  - sum_T(OWD > mean_delay)

       skew_est = sum_MT(skew_base_T)/num_MT(OWD)
```

This calculation requires slightly more state, since an implementation will need to maintain two cyclic buffers storing skew_base_T and num_T(OWD) respectively to manage the rolling summations (note only one cyclic buffer is needed for the calculation of skew_est outlined previously).

3.4. Reducing lag and Improving Responsiveness

Measurement based shared bottleneck detection makes decisions in the present based on what has been measured in the past. This means that there is always a lag in responding to changing conditions. This mechanism is based on summary statistics taken over (N*T) seconds. This mechanism can be made more responsive to changing conditions by:

1. Reducing N and/or M -- but at the expense of less accurate metrics, and/or

2. Exploiting the fact that more recent measurements are more valuable than older measurements and weighting them accordingly.

Although more recent measurements are more valuable, older measurements are still needed to gain an accurate estimate of the distribution descriptor we are measuring. Unfortunately, the simple exponentially weighted moving average weights drop off too quickly for our requirements and have an infinite tail. A simple linearly declining weighted moving average also does not provide enough weight to the most recent measurements. We propose a piecewise linear distribution of weights, such that the first section (samples 1:F) is flat as in a simple moving average, and the second section (samples F+1:M) is linearly declining weights to the end of the averaging window. We choose integer weights, which allows incremental calculation without introducing rounding errors.

3.4.1. Improving the response of the skewness estimate

The weighted moving average for skew_est, based on skew_est in Section 3.3.3, can be calculated as follows:

```
skew_est = ((M-F+1)*sum(skew_base_T(1:F))

                + sum([(M-F):1].*skew_base_T(F+1:M)))

           / ((M-F+1)*sum(numsampT(1:F))

                + sum([(M-F):1].*numsampT(F+1:M)))
```

where numsampT is an array of the number of OWD samples in each T (ie num_T(OWD)), and numsampT(1) is the most recent; skew_base_T(1) is the most recent calculation of skew_base_T; 1:F refers to the integer values 1 through to F, and [(M-F):1] refers to an array of the integer values (M-F) declining through to 1; and ".*" is the array scalar dot product operator.

3.4.2. Improving the response of the variance estimate

The weighted moving average for var_est can be calculated as follows:

```
var_est = ((M-F+1)*sum(PDV(1:F)) + sum([(M-F):1].*PDV(F+1:M)))

          / (F*(M-F+1) + sum([(M-F):1])
```

where 1:F refers to the integer values 1 through to F, and [(M-F):1] refers to an array of the integer values (M-F) declining through to 1; and ".*" is the array scalar dot product operator. When removing oscillation noise (see Section 3.3.1) this calculation must be adjusted to allow for invalid PDV records.

4. Measuring OWD

This section discusses the OWD measurements required for this algorithm to detect shared bottlenecks.

The SBD mechanism described in this draft relies on differences between OWD measurements to avoid the practical problems with measuring absolute OWD (see [Hayes-LCN14] section IIIC). Since all summary statistics are relative to the mean OWD and sender/receiver clock offsets should be approximately constant over the measurement periods, the offset is subtracted out in the calculation.

4.1. Time stamp resolution

The SBD mechanism requires timing information precise enough to be able to make comparisons. As a rule of thumb, the time resolution should be less than one hundredth of a typical path's range of delays. In general, the lower the time resolution, the more care that needs to be taken to ensure rounding errors do not bias the skewness calculation.

Typical RTP media flows use sub-millisecond timers, which should be adequate in most situations.

5. Acknowledgements

This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The security considerations of RFC 3550 [RFC3550], RFC 4585 [RFC4585], and RFC 5124 [RFC5124] are expected to apply.

Non-authenticated RTCP packets carrying shared bottleneck indications and summary statistics could allow attackers to alter the bottleneck sharing characteristics for private gain or disruption of other parties communication.

8. Change history

Changes made to this document:

01->02 : New section describing improvements to the key metric calculations that help to remove noise, bias, and reduce lag. Some revisions to the notation to make it clearer. Some tightening of the thresholds.

00->01 : Revisions to terminology for clarity

9. References

9.1. Normative References

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

9.2. Informative References

[Hayes-LCN14]
Hayes, D., Ferlin, S., and M. Welzl, "Practical Passive Shared Bottleneck Detection using Shape Summary Statistics", Proc. the IEEE Local Computer Networks (LCN) p150-158, September 2014, <http://heim.ifi.uio.no/davihay/hayes14__pract_passiv_shared_bottl_detec-abstract.html>.

[I-D.welzl-rmcat-coupled-cc]
Welzl, M., Islam, S., and S. Gjessing, "Coupled congestion control for RTP media", draft-welzl-rmcat-coupled-cc-04 (work in progress), October 2014.

[ITU-Y1540]
ITU-T, "Internet Protocol Data Communication Service - IP Packet Transfer and Availability Performance Parameters", Series Y: Global Information Infrastructure, Internet Protocol Aspects and Next-Generation Networks , March 2011, <http://www.itu.int/rec/T-REC-Y.1540-201103-I/en>.

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

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control

Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", RFC 5481, March 2009.

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

Authors' Addresses

David Hayes (editor)
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 2284 5566
Email: davihay@ifi.uio.no

Simone Ferlin
Simula Research Laboratory
P.O.Box 134
Lysaker, 1325
Norway

Phone: +47 4072 0702
Email: ferlin@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 2285 2420
Email: michawe@ifi.uio.no

RMCAT WG   I. Johansson
Internet-Draft   Z. Sarker
Intended status: Informational   Ericsson AB
Expires: September 3, 2015   March 2, 2015

Self-Clocked Rate Adaptation for Multimedia
draft-johansson-rmcat-scream-cc-05

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Congestion in the internet is a reality and applications that are deployed in the internet must have congestion control schemes in place not only for the robustness of the service that it provides but also to ensure the function of the currently deployed internet. As the interactive realtime communication imposes a great deal of requirements on the transport, a robust, efficient rate adaptation for all access types is considered as an important part of interactive realtime communications as the transmission channel bandwidth may vary over time. Wireless access such as LTE, which is an integral part of the current internet, increases the importance of rate adaptation as the channel bandwidth of a default LTE bearer [QoS-3GPP] can change considerably in a very short time frame. Thus a rate adaptation solution for interactive realtime media, such as WebRTC, must be both quick and be able to operate over a large span in available channel bandwidth. This memo describes a solution,named SCReAM, that is based on the self-clocking principle of TCP and uses techniques similar to what is used in a new delay based rate adaptation algorithm, LEDBAT [RFC6817]. Because neither TCP nor LEDBAT was designed for interactive realtime media, a few extra features are needed to make the concept work well within this context. This memo describes these extra features.

1.1. Wireless (LTE) access properties

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

Unlike wireline bottlenecks with large statistical multiplexing it is not possible to try to maintain a given bitrate when congestion is detected with the hope that other flows will yield, this because there are generally few other flows competing for the same bottleneck. Each user gets its own variable throughput bottleneck, where the throughput depends on factors like channel quality, network load and historical throughput. The bottom line is, if the throughput drops, the sender has no other option than to reduce the bitrate. In addition, the grace time, i.e. allowed reaction time from the time that the congestion is detected until a reaction in terms of a rate reduction is effected, is generally very short, in the order of one RTT (Round Trip Time).

2. Terminology

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

3. Overview of SCReAM Algorithm

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

The packet conservation principle is realized by including an indication of the highest received sequence number in the feedback, see Section 5, from the receiver back to the sender, the sender keeps a list of transmitted packets and their respective sizes. This information is then used to determine how many bytes can be transmitted. A congestion window puts an upper limit on how many bytes can be in flight, i.e. transmitted but not yet acknowledged. The congestion window is determined in a way similar to LEDBAT [RFC6817]. This ensures that the e2e latency is kept low. The basic functionality is quite simple, there are however a few steps to take to make the concept work with conversational media. These will be briefly described in sections Section 3.1 to Section 3.3.

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

3.1. Congestion Control

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

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

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

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

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

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

3.2. Transmission Scheduling

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

3.3. Media Rate Control

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

The reaction to reduced throughput must be prompt in order to avoid getting too much data queued up in the RTP packet queues. The media bitrate is decreased if the RTP queue size exceeds a threshold.

In cases where the sender frame queues increase rapidly such as the case of a RAT (Radio Access Type) handover it may be necessary to implement additional actions, such as discarding of encoded video frames or frame skipping in order to ensure that the RTP queues are drained quickly. Frame skipping means that the frame rate is temporarily reduced. Discarding of old video frames is a more efficient way to reduce media latency than frame skipping but it comes with a requirement to repair codec state, frame skipping is thus to prefer as a first remedy. Frame skipping is described as an optional to implement feature in this specification.

4. Detailed Description of SCReAM

4.1. SCReAM Sender

This section describes the sender side algorithm in more detail. It is split between the network congestion control and the video rate adaptation.

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

```
        ----------------------------    -----------------------------
        |       Video encoder      |    |        Video encoder       |
        ----------------------------    -----------------------------
         ^                |       ^      ^                 |       ^
      (1)|             (2)|    (3)|   (1)|              (2)|    (3)|
         |               RTP      |      |                RTP      |
         |                V       |      |                 V       |
         |          ------------- |      |           ------------- |
    -----------     |           |--  -----------     |           |--
    | Rate    | (4) |   Queue   |    | Rate    | (4) |   Queue   |
    | control |<----|           |    | control |<----|           |
    |         |     |RTP packets|    |         |     |RTP packets|
    -----------     |           |    -----------     |           |
                    -------------                    -------------
                          |                                |
                          ---------------     --------------
                                     (5)|     |(5)
                                       RTP   RTP
                                        |     |
                                        v     v
           --------------          ----------------
           |  Network   |    (8)   | Transmission |
           | congestion |<-------->|   scheduler  |
           |  control   |          |              |
           --------------          ----------------
                ^                         |
                |         (7)             |(6)
                ---------RTCP----------  RTP
                                      |   |
                                      |   v
                                  -------------
                                  |    UDP    |
                                  |  socket   |
                                  -------------
```

Figure 1: SCReAM sender functional view

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

4.1.1. Constants and Parameter values

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

An init value [] indicates an empty array.

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

Table 1: Constants

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

	interval [s], updated when RTP packet transmitted	
age_vec	A vector of the last 20 RTP packet queue delay samples	[]
frame_skip_intensity	Indicates the intensity of the frame skips	0.0
since_last_frame_skip	Number of video frames since the last skip	0
consecutive_frame_skips	Number of consecutive frame skips	0
target_bitrate	Video target bitrate [bps]	TARGET_BITRATE_MIN
target_bitrate_i	Video target bitrate inflection point i.e. the last known highest target_bitrate during fast start. Used to limit bitrate increase close to the last know congestion point	1
rate_transmit	Measured transmit bitrate [bps]	0.0
rate_acked	Measured throughput based on received acknowledgements [bps]	0.0
s_rtt	Smoothed RTT [s], computed similar to method depicted in [RFC6298]	0.0
rtp_queue_size	Size of RTP packets in queue [bits]	0
rtp_size	Size of the last transmitted RTP packets [byte]	0
frame_skip	Skip encoding of video frame if	false

	true	

Table 2: State variables

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

Table 3: Local temporary variables

4.1.2. Network congestion control

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

- o Computation of congestion window at the sender: Gives an upper limit to the number of bytes in flight i.e. how many bytes that have been transmitted but not yet acknowledged.
- o Transmission scheduling at the sender: RTP packets are transmitted if allowed by the relation between the number of bytes in flight and the congestion window. This is controlled by the send window.

Unlike TCP, SCReAM is not a byte oriented protocol, rather it is an RTP packet oriented protocol. Thus it keeps a list of transmitted RTP packets and their respective sending times (wall-clock time). The feedback indicates the highest received RTP sequence number and a timestamp (wall-clock time) when it was received. In addition, an ACK list is included to make it possible to determine lost packets.

4.1.2.1. Congestion window update

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

SCReAM uses the terminology "Bytes in flight (bytes_in_flight)" which is computed as the sum of the sizes of the RTP packets ranging from the RTP packet most recently transmitted down to but not including the acknowledged packet with the highest sequence number. As an example: If RTP packet was sequence number SN with transmitted and the last ACK indicated SN-5 as the highest received sequence number then bytes in flight is computed as the sum of the size of RTP packets with sequence number SN-4, SN-3, SN-2, SN-1 and SN.

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

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

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

bytes_newly_acked is updated.

The OWD fraction and an average of it are computed as

owd_fraction = owd/owd_target

owd_fraction_avg = 0.9* owd_fraction_avg + 0.1* owd_fraction

The OWD fraction is sampled every 50ms and the last 20 samples are stored in a vector (owd_fraction_hist). This vector is used in the computation of an OWD trend that gives a value between 0.0 and 1.0

depending on how close to congestion it is. The OWD trend is calculated as follows

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

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

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

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

The owd_trend is utilized in the media rate control and to determine when to exit slow start.

An off target value is computed as

off_target = (owd_target - owd) / owd_target

A temporal variable is scl_i is computed as

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

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

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

On loss event:

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

cwnd_i = cwnd

cwnd = max(min_cwnd,cwnd*BETA)

otherwise the CWND update continues

---

in_fast_start = true:

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

cwnd = cwnd + bytes_newly_acked*scl_i

---

in_fast_start = false:

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

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

The equation above limits the gain when near congestion is detected

gain *= scl_i

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

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

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

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

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

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

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

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

---

Resume fast start:

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

Competing flows compensation, adjustment of owd_target:

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

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

Final CWND adjustment step:

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

```
cwnd = min(cwnd, max_bytes_in_flight*MAX_BYTES_IN_FLIGHT_HEAD_ROOM)
```

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

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

```
cwnd = max(cwnd, MIN_CWND)
```

4.1.2.2. Transmission scheduling

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

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

- o Reverse (feedback) path congestion: Especially in transport over buffer-bloated networks, the one way delay in the reverse direction may jump due to congestion. The effect of this is that the acknowledgements are delayed with the result that the self-clocking is temporarily halted, even though the forward path is not congested.

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

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

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

- o If owd <= owd_target:
  A helper variable
  x_cwnd=1.0+BYTES_IN_FLIGHT_SLACK*max(0.0,
  min(1.0,1.0-owd_trend/0.5))/100.0
  is computed. The send window is computed as
  send_wnd = max(cwnd*x_cwnd, cwnd+mss)-bytes_in_flight
  This gives a slack that reduces as congestion increases, BYTES_IN_FLIGHT_SLACK is a maximum allowed slack in percent. A large value increases the robustness to bitrate variations in the source and congested feedback channel issues. The possible drawback is increased delay or packet loss when forward path congestion occur.

4.1.3. Video rate control

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

A variable target_bitrate is adjusted depending on the congestion state. The target bitrate can vary between a minimum value (target_bitrate_min) and a maximum value (target_bitrate_max).

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

- o rate_transmit: The measured transmit bitrate

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

Both estimates are updated every 200ms.

The current throughput current_rate is computed as the maximum value of rate_transmit and rate_acked. The rationale behind the use of rate_acked in addition to rate_transmit is that rate_transmit is affected also by the amount of data that is available to transmit, thus a lack of data to transmit can be seen as reduced throughput that may itself cause an unnecessary rate reduction. To overcome this shortcoming; rate_acked is used as well. This gives a more stable throughput estimate.

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

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

On loss event:

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

target_bitrate_i = target_bitrate

target_bitrate = max(BETA_R* target_bitrate, TARGET_BITRATE_MIN)

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

---

in_fast_start = true:

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

```
scl_i = (target_bitrate - target_bitrate_i)/ target_bitrate_i

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

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

target_bitrate += increment
```

target_bitrate is reduced further if congestion is detected.

```
target_bitrate *= (1.0- PRE_CONGESTION_GUARD*owd_trend)

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

---

in_fast_start = false:

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

A pre-congestion indicator is computed as

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

pre_congestion += owd_trend
```

The target bitrate is computed as

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

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

4.2. SCReAM Receiver

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

One set of state variables are maintained per stream.

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

Table 4: State variables

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

5. Feedback Message

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

- o Highest received RTP sequence number : The highest received RTP sequence number for the given SSRC
- o n_lost : Ackumulated number of lost RTP packets for the given SSRC
- o Timestamp: A timestamp value indicating when the last packet was received which makes it possible to compute the one way (extra) delay (OWD).
- o n_ECN : Ackumulated number of ECN-CE marked RTP packets for the given SSRC
- o Source quench bit (Q): Makes it possible to request the sender to reduce its congestion window. This is useful if WebRTC media is received from many hosts and it becomes necessary to balance the bitrates between the streams.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|   FMT   |       PT      |          length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  SSRC of packet sender                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  SSRC of media source                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Highest recv. seq. nr. (16b)  |    n_lost     |   n_ECN       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Timestamp (32bits)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Q|                Reserved for future use                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Transport layer feedback message

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

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

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

6. Additional features

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

6.1. Packet pacing

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

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

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

t_pace = rtp_size * 8 / pace_bitrate

rtp_size is the size of the last transmitted RTP packet

6.2. Frame skipping

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

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

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

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

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

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

The change in age_avg is computed as

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

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

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

- Otherwise frame skip intensity is set to zero

The skip_frame flag is set depending on three variables

- frame_skip_intensity

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

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

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

- age_vec(n) > 0.2 && consecutive_frame_skips < 5

- frame_skip_intensity < 0.5 && since_last_frame_skip >= 1.0/frame_skip_intensity

- frame_skip_intensity >= 0.5 && consecutive_frame_skips < (frame_skip_intensity -0.5)*10

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

6.3. Q-bit semantics (source quench)

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

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

- n = Number of received RTCP feedback messages in one RTT

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

The new congestion window is then expressed as:

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

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

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

7. Discussion

This section covers a few open discussion points

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

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

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

8. Conclusion

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

9. Open issues

A list of open issues.

- o Describe how clock drift compensation is done

- o Describe how FEC overhead is accounted for in target_bitrate computation

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

10. Source code

Source code for SCReAM is available in two formats :

- o C++ code, that is apt for experimentation. The code maitained as Visual Studio project. This code can possibly be included in simulators such as NS3. Avaliable at https://github.com/EricssonResearch/scream

- o OpenWebRTC implementation : Work in progress, see http://www.openwebrtc.io/ for information about the OpenWebRTC project

11. 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 Haekansson, Erlendur Karlsson, Daniel Lindstroem, Mats Nordberg, Jonathan Samuelsson, Rickard Sjoeberg, Robert Swain, Magnus Westerlund, Stefan Aelund.

12. IANA Considerations

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

13. 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.

14. Change history

A list of changes:

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

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

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

- o -01 to -02 : Updated GCC simulation results o -00 to -01 : Fixed a few bugs in example code

15. References

15.1. Normative References

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

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

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

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

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

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

15.2. Informative References

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

[I-D.draft-sarker-rmcat-cellular-eval-test-cases]
Sarker, Z., "Evaluation Test Cases for Interactive Real-Time Media over Cellular Networks", <http://www.ietf.org/id/draft-sarker-rmcat-cellular-eval-test-cases-00.txt>.

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

[I-D.ietf-tcpm-newcwv]
Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to support Rate-Limited Traffic", draft-ietf-tcpm-newcwv-08 (work in progress), February 2015.

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

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

Authors' Addresses

Ingemar Johansson
Ericsson AB
Laboratoriegraend 11
Luleae 977 53
Sweden

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

Zaheduzzaman Sarker
Ericsson AB
Laboratoriegraend 11
Luleae 977 53
Sweden

Phone: +46 761153743
Email: zaheduzzaman.sarker@ericsson.com

M. Welzl
S. Islam
S. Gjessing
University of Oslo
October 24, 2014

Coupled congestion control for RTP media
draft-welzl-rmcat-coupled-cc-04

Abstract

When multiple congestion controlled RTP sessions traverse the same network bottleneck, it can be beneficial to combine their controls such that the total on-the-wire behavior is improved. This document describes such a method for flows that have the same sender, in a way that is as flexible and simple as possible while minimizing the amount of changes needed to existing RTP applications.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

When there is enough data to send, a congestion controller must increase its sending rate until the path's capacity has been reached; depending on the controller, sometimes the rate is increased further, until packets are ECN-marked or dropped. This process inevitably creates undesirable queuing delay -- an effect that is amplified when multiple congestion controlled connections traverse the same network bottleneck. When such connections originate from the same host, it would therefore be ideal to use only one single sender-side congestion controller which determines the overall allowed sending rate, and then use a local scheduler to assign a proportion of this rate to each RTP session. This way, priorities could also be implemented quite easily, as a function of the scheduler; honoring user-specified priorities is, for example, required by rtcweb [rtcweb-usecases].

The Congestion Manager (CM) [RFC3124] provides a single congestion controller with a scheduling function just as described above. It is hard to implement because it requires an additional congestion controller and removes all per-connection congestion control functionality, which is quite a significant change to existing RTP based applications. This document presents a method that is easier to implement than the CM and also requires less significant changes to existing RTP based applications. It attempts to roughly approximate the CM behavior by sharing information between existing congestion controllers, akin to "Ensemble Sharing" in [RFC2140].

2. Definitions

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 RFC 2119 [RFC2119].

Available Bandwidth:
: The available bandwidth is the nominal link capacity minus the amount of traffic that traversed the link during a certain time interval, divided by that time interval.

Bottleneck:
: The first link with the smallest available bandwidth along the path between a sender and receiver.

Flow:
: A flow is the entity that congestion control is operating on. It could, for example, be a transport layer connection, an RTP session, or a subsession that is multiplexed onto a single RTP

session together with other subsessions.

Flow Group Identifier (FGI):
A unique identifier for each subset of flows that is limited by a common bottleneck.

Flow State Exchange (FSE):
The entity that maintains information that is exchanged between flows.

Flow Group (FG):
A group of flows having the same FGI.

Shared Bottleneck Detection (SBD):
The entity that determines which flows traverse the same bottleneck in the network, or the process of doing so.

3. Limitations

Sender-side only:
Coupled congestion control as described here only operates inside a single host on the sender side. This is because, irrespective of where the major decisions for congestion control are taken, the sender of a flow needs to eventually decide the transmission rate. Additionally, the necessary information about how much data an application can currently send on a flow is typically only available at the sender side, making the sender an obvious choice for placement of the elements and mechanisms described here.

When implementing a sender-side change to a congestion control mechanism such as TFRC [RFC5348], where receiver-side calculations make assumptions about the rate of the sender, the receiver also needs to be updated accordingly. Flows that have different senders but the same receiver, or different senders and different receivers can also share a bottleneck; such scenarios have been omitted for simplicity, and could be incorporated in future versions of this document. Note that limiting the number of flows on which coupled congestion control operates merely limits the benefits derived from the mechanism.

Shared bottlenecks do not change quickly:
As per the definition above, a bottleneck depends on cross traffic, and since such traffic can heavily fluctuate, bottlenecks can change at a high frequency (e.g., there can be oscillation between two or more links). This means that, when

flows are partially routed along different paths, they may quickly change between sharing and not sharing a bottleneck. For simplicity, here it is assumed that a shared bottleneck is valid for a time interval that is significantly longer than the interval at which congestion controllers operate. Note that, for the only SBD mechanism defined in this document (multiplexing on the same five-tuple), the notion of a shared bottleneck stays correct even in the presence of fast traffic fluctuations: since all flows that are assumed to share a bottleneck are routed in the same way, if the bottleneck changes, it will still be shared.

4. Architectural overview

Figure 1 shows the elements of the architecture for coupled congestion control: the Flow State Exchange (FSE), Shared Bottleneck Detection (SBD) and Flows. The FSE is a storage element that can be implemented in two ways: active and passive. In the active version, it initiates communication with flows and SBD. However, in the passive version, it does not actively initiate communication with flows and SBD; its only active role is internal state maintenance (e.g., an implementation could use soft state to remove a flow's data after long periods of inactivity). Every time a flow's congestion control mechanism would normally update its sending rate, the flow instead updates information in the FSE and performs a query on the FSE, leading to a sending rate that can be different from what the congestion controller originally determined. Using information about/from the currently active flows, SBD updates the FSE with the correct Flow State Identifiers (FSIs).

```
-------  <---  Flow 1
| FSE |  <---  Flow 2 ..
-------  <---  .. Flow N
   ^
   |              |
-------           |
| SBD |  <-------|
-------
```

Figure 1: Coupled congestion control architecture

Since everything shown in Figure 1 is assumed to operate on a single host (the sender) only, this document only describes aspects that have an influence on the resulting on-the-wire behavior. It does, for instance, not define how many bits must be used to represent FSIs, or in which way the entities communicate. Implementations can take various forms: for instance, all the elements in the figure could be implemented within a single application, thereby operating on flows generated by that application only. Another alternative could be to implement both the FSE and SBD together in a separate process which different applications communicate with via some form of Inter-Process Communication (IPC). Such an implementation would extend the scope to flows generated by multiple applications. The FSE and SBD could also be included in the Operating System kernel.

5. Roles

This section gives an overview of the roles of the elements of coupled congestion control, and provides an example of how coupled congestion control can operate.

5.1. SBD

SBD uses knowledge about the flows to determine which flows belong in the same Flow Group (FG), and assigns FGIs accordingly. This knowledge can be derived in three basic ways:

1. From multiplexing: it can be based on the simple assumption that packets sharing the same five-tuple (IP source and destination address, protocol, and transport layer port number pair) and having the same Differentiated Services Code Point (DSCP) in the IP header are typically treated in the same way along the path. The latter method is the only one specified in this document: SBD MAY consider all flows that use the same five-tuple and DSCP to belong to the same FG. This classification applies to certain tunnels, or RTP flows that are multiplexed over one transport (cf. [transport-multiplex]). In one way or another, such multiplexing will probably be recommended for use with rtcweb [rtcweb-rtp-usage].

2. Via configuration: e.g. by assuming that a common wireless uplink is also a shared bottleneck.

3. From measurements: e.g. by considering correlations among measured delay and loss as an indication of a shared bottleneck.

The methods above have some essential trade-offs: e.g., multiplexing is a completely reliable measure, however it is limited in scope to two end points (i.e., it cannot be applied to couple congestion controllers of one sender talking to multiple receivers). A measurement-based SBD mechanism is described in [sbd]. Measurements can never be 100% reliable, in particular because they are based on the past but applying coupled congestion control means to make an assumption about the future; it is therefore recommended to implement cautionary measures, e.g. by disabling coupled congestion control if enabling it causes a significant increase in delay and/or packet loss. Measurements also take time, which entails a certain delay for turning on coupling (refer to [sbd] for details).

5.2. FSE

The FSE contains a list of all flows that have registered with it. For each flow, it stores the following:

- a unique flow number to identify the flow
- the FGI of the FG that it belongs to (based on the definitions in this document, a flow has only one bottleneck, and can therefore be in only one FG)
- a priority P, which here is assumed to be represented as a floating point number in the range from 0.1 (unimportant) to 1 (very important). A negative value is used to indicate that a flow has terminated
- The rate used by the flow in bits per second, FSE_R.

The FSE can operate on window-based as well as rate-based congestion controllers (TEMPORARY NOTE: and probably -- not yet tested -- combinations thereof, with calculations to convert from one to the other). In case of a window-based controller, FSE_R is a window, and all the text below should be considered to refer to window, not rates.

In the FSE, each FG contains one static variable S_CR which is meant to be the sum of the calculated rates of all flows in the same FG (including the flow itself). This value is used to calculate the sending rate.

The information listed here is enough to implement the sample flow algorithm given below. FSE implementations could easily be extended to store, e.g., a flow’s current sending rate for statistics gathering or future potential optimizations.

5.3. Flows

Flows register themselves with SBD and FSE when they start, deregister from the FSE when they stop, and carry out an UPDATE function call every time their congestion controller calculates a new sending rate. Via UPDATE, they provide the newly calculated rate and the desired rate (less than the calculated rate in case of application-limited flows, the same otherwise).

Below, two example algorithms are described. While other algorithms could be used instead, the same algorithm must be applied to all flows.

5.3.1. Example algorithm 1 - Active FSE

This algorithm was designed to be the simplest possible method to assign rates according to the priorities of flows. Simulations results in [fse] indicate that it does however not significantly reduce queuing delay and packet loss.

(1) When a flow f starts, it registers itself with SBD and the FSE. FSE_R is initialized with the congestion controller's initial rate. SBD will assign the correct FGI. When a flow is assigned an FGI, it adds its FSE_R to S_CR.

(2) When a flow f stops, its entry is removed from the list.

(3) Every time the congestion controller of the flow f determines a new sending rate CC_R, the flow calls UPDATE, which carries out the tasks listed below to derive the new sending rates for all the flows in the FG. A flow's UPDATE function uses a local (i.e. per-flow) temporary variable S_P, which is the sum of all the priorities.

    (a) It updates S_CR.

    ```
    S_CR = S_CR + CC_R - FSE_R(f)
    ```

    (b) It calculates the sum of all the priorities, S_P.

    ```
    S_P = 0
    for all flows i in FG do
        S_P = S_P + P(i)
    end for
    ```

    (c) It calculates the sending rates for all the flows in an FG and distributes them.

    ```
    for all flows i in FG do
        FSE_R(i) = (P(i)*S_CR)/S_P
    ```

```
        send FSE_R(i) to the flow i
    end for
```

5.3.2. Example algorithm 2 - Conservative Active FSE

This algorithm extends algorithm 1 to conservatively emulate the behavior of a single flow by proportionally reducing the aggregate rate on congestion. Simulations results in [fse] indicate that it can significantly reduce queuing delay and packet loss.

(1) When a flow f starts, it registers itself with SBD and the FSE. FSE_R is initialized with the congestion controller’s initial rate. SBD will assign the correct FGI. When a flow is assigned an FGI, it adds its FSE_R to S_CR.

(2) When a flow f stops, its entry is removed from the list.

(3) Every time the congestion controller of the flow f determines a new sending rate CC_R, the flow calls UPDATE, which carries out the tasks listed below to derive the new sending rates for all the flows in the FG. A flow’s UPDATE function uses a local (i.e. per-flow) temporary variable S_P, which is the sum of all the priorities, and a local variable DELTA, which is used to calculate the difference between CC_R and the previously stored FSE_R. To prevent flows from either ignoring congestion or overreacting, a timer keeps them from changing their rates immediately after the common rate reduction that follows a congestion event. This timer is set to 2 RTTs of the flow that experienced congestion because it is assumed that a congestion event can persist for up to one RTT of that flow, with another RTT added to compensate for fluctuations in the measured RTT value.

   (a) It updates S_CR based on DELTA.

```
if Timer has expired or not set then
  DELTA = CC_R - FSE_R(f)
  if DELTA < 0 then  // Reduce S_CR proportionally
    S_CR = S_CR * CC_R / FSE_R(f)
    Set Timer for 2 RTTs
  else
    S_CR = S_CR + DELTA
  end if
 end if
```

(b) It calculates the sum of all the priorities, S_P.

```
S_P = 0
for all flows i in FG do
    S_P = S_P + P(i)
end for
```

(c) It calculates the sending rates for all the flows in an FG and distributes them.

```
for all flows i in FG do
    FSE_R(i) = (P(i)*S_CR)/S_P
    send FSE_R(i) to the flow i
end for
```

6. Acknowledgements

This document has benefitted from discussions with and feedback from David Hayes, Andreas Petlund, and David Ros (who also gave the FSE its name).

This work was partially funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

In scenarios where the architecture described in this document is applied across applications, various cheating possibilities arise: e.g., supporting wrong values for the calculated rate, the desired rate, or the priority of a flow. In the worst case, such cheating could either prevent other flows from sending or make them send at a rate that is unreasonably large. The end result would be unfair behavior at the network bottleneck, akin to what could be achieved with any UDP based application. Hence, since this is no worse than UDP in general, there seems to be no significant harm in using this in the absence of UDP rate limiters.

In the case of a single-user system, it should also be in the interest of any application programmer to give the user the best possible experience by using reasonable flow priorities or even letting the user choose them. In a multi-user system, this interest may not be given, and one could imagine the worst case of an "arms race" situation, where applications end up setting their priorities to the maximum value. If all applications do this, the end result is a fair allocation in which the priority mechanism is implicitly eliminated, and no major harm is done.

9. References

9.1. Normative References

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

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

9.2. Informative References

[fse] Islam, S., Welzl, M., Gjessing, S., and N. Khademi, "Coupled Congestion Control for RTP Media", ACM SIGCOMM Capacity Sharing Workshop (CSWS 2014); extended version available as a technical report from http://safiquli.at.ifi.uio.no/paper/fse-tech-report.pdf , 2014.

[rtcweb-rtp-usage] Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-18.txt (work in progress), October 2014.

[rtcweb-usecases] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-14.txt (work in progress), February 2014.

[sbd] Hayes, D., Ferlin, S., and M. Welzl, "Shared Bottleneck Detection for Coupled Congestion Control for RTP Media", draft-hayes-rmcat-sbd-00.txt (work in progress), October 2014.

[transport-multiplex] Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing-07.txt (work in progress), October 2013.

Appendix A. Example algorithm - Passive FSE

Active algorithms calculate the rates for all the flows in the FG and actively distribute them. In a passive algorithm, UPDATE returns a rate that should be used instead of the rate that the congestion controller has determined. This can make a passive algorithm easier to implement; however, the resulting dynamics are not fully understood. The algorithm described below is to be considered as highly experimental and did not perform as well as the active variants in simulations.

This passive version of the FSE stores the following information in addition to the variables described in Section 5.2:

o The desired rate DR. This can be smaller than the calculated rate if the application feeding into the flow has less data to send than the congestion controller would allow. In case of a bulk transfer, DR must be set to CC_R received from the flow's congestion module.

The passive version of the FSE contains one static variable per FG called TLO (Total Leftover Rate -- used to let a flow 'take' bandwidth from application-limited or terminated flows) which is initialized to 0. For the passive version, S_CR is limited to increase or decrease as conservatively as a flow's congestion controller decides in order to prohibit sudden rate jumps.

(1) When a flow f starts, it registers itself with SBD and the FSE. FSE_R and DR are initialized with the congestion controller's initial rate. SBD will assign the correct FGI. When a flow is assigned an FGI, it adds its FSE_R to S_CR.

(2) When a flow f stops, it sets its DR to 0 and sets P to -1.

(3) Every time the congestion controller of the flow f determines a new sending rate CC_R, assuming the flow's new desired rate new_DR to be "infinity" in case of a bulk data transfer with an unknown maximum rate, the flow calls UPDATE, which carries out the tasks listed below to derive the flow's new sending rate, Rate. A flow's UPDATE function uses a few local (i.e. per-flow) temporary variables, which are all initialized to 0: DELTA, new_S_CR and S_P.

(a) For all the flows in its FG (including itself), it calculates the sum of all the calculated rates, new_S_CR. Then it calculates the difference between FSE_R(f) and CC_R, DELTA.

```
for all flows i in FG do
    new_S_CR = new_S_CR + FSE_R(i)
end for
DELTA =  CC_R - FSE_R(f)
```

(b) It updates S_CR, FSE_R(f) and DR(f).

```
FSE_R(f) = CC_R
if DELTA > 0 then  // the flow's rate has increased
    S_CR = S_CR + DELTA
else if DELTA < 0 then
    S_CR = new_S_CR + DELTA
end if
DR(f) = min(new_DR,FSE_R(f))
```

(c) It calculates the leftover rate TLO, removes the terminated flows from the FSE and calculates the sum of all the priorities, S_P.

```
for all flows i in FG do
   if P(i)<0 then
      delete flow
   else
      S_P = S_P + P(i)
   end if
end for
if DR(f) < FSE_R(f) then
   TLO = TLO + (P(f)/S_P) * S_CR - DR(f))
end if
```

(d) It calculates the sending rate, Rate.

```
Rate = min(new_DR, (P(f)*S_CR)/S_P + TLO)

if Rate != new_DR and TLO > 0 then
    TLO = 0  // f has 'taken' TLO
end if
```

(e) It updates DR(f) and FSE_R(f) with Rate.

```
if Rate > DR(f) then
    DR(f) = Rate
end if
FSE_R(f)  = Rate
```

The goals of the flow algorithm are to achieve prioritization, improve network utilization in the face of application-limited flows, and impose limits on the increase behavior such that the negative impact of multiple flows trying to increase their rate together is minimized. It does that by assigning a flow a sending rate that may not be what the flow's congestion controller expected. It therefore builds on the assumption that no significant inefficiencies arise from temporary application-limited behavior or from quickly jumping to a rate that is higher than the congestion controller intended. How problematic these issues really are depends on the controllers in use and requires careful per-controller experimentation. The coupled congestion control mechanism described here also does not require all controllers to be equal; effects of heterogeneous controllers, or homogeneous controllers being in different states, are also subject to experimentation.

This algorithm gives all the leftover rate of application-limited flows to the first flow that updates its sending rate, provided that this flow needs it all (otherwise, its own leftover rate can be taken by the next flow that updates its rate). Other policies could be applied, e.g. to divide the leftover rate of a flow equally among all other flows in the FGI.

A.1. Example operation (passive)

In order to illustrate the operation of the passive coupled congestion control algorithm, this section presents a toy example of two flows that use it. Let us assume that both flows traverse a common 10 Mbit/s bottleneck and use a simplistic congestion controller that starts out with 1 Mbit/s, increases its rate by 1 Mbit/s in the absence of congestion and decreases it by 2 Mbit/s in the presence of congestion. For simplicity, flows are assumed to always operate in a round-robin fashion. Rate numbers below without units are assumed to be in Mbit/s. For illustration purposes, the actual sending rate is also shown for every flow in FSE diagrams even though it is not really stored in the FSE.

Flow #1 begins. It is a bulk data transfer and considers itself to have top priority. This is the FSE after the flow algorithm's step 1:

#	FGI	P	FSE_R	DR	Rate
1	1	1	1	1	1

S_CR = 1, TLO = 0

Its congestion controller gradually increases its rate. Eventually, at some point, the FSE should look like this:

#	FGI	P	FSE_R	DR	Rate
1	1	1	10	10	10

S_CR = 10, TLO = 0

Now another flow joins. It is also a bulk data transfer, and has a lower priority (0.5):

#	FGI	P	FSE_R	DR	Rate
1	1	1	10	10	10
2	1	0.5	1	1	1

S_CR = 11, TLO = 0

Now assume that the first flow updates its rate to 8, because the total sending rate of 11 exceeds the total capacity. Let us take a closer look at what happens in step 3 of the flow algorithm.

CC_R = 8. new_DR = infinity.
3 a) new_S_CR = 11; DELTA = 8 - 10 = -2.
3 b) FSE_Rf) = 8. DELTA is negative, hence S_CR = 9;
     DR(f) = 8.
3 c) S_P = 1.5.
3 d) new sending rate = min(infinity, 1/1.5 * 9 + 0) = 6.
3 e) FSE_R(f) = 6.

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	6	8	6
2	1	0.5	1	1	1

S_CR = 9, TLO = 0

The effect is that flow #1 is sending with 6 Mbit/s instead of the 8 Mbit/s that the congestion controller derived. Let us now assume that flow #2 updates its rate. Its congestion controller detects that the network is not fully saturated (the actual total sending rate is 6+1=7) and increases its rate.

CC_R=2. new_DR = infinity.
3 a) new_S_CR = 7; DELTA = 2 - 1 = 1.
3 b) FSE_R(f) = 2. DELTA is positive, hence S_CR = 9 + 1 = 10;
     DR(f) = 2.
3 c) S_P = 1.5.
3 d) new sending rate = min(infinity, 0.5/1.5 * 10 + 0) = 3.33.
3 e) DR(f) = FSE_R(f) = 3.33.

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	6	8	6
2	1	0.5	3.33	3.33	3.33

S_CR = 10, TLO = 0

The effect is that flow #2 is now sending with 3.33 Mbit/s, which is close to half of the rate of flow #1 and leads to a total utilization of 6(#1) + 3.33(#2) = 9.33 Mbit/s. Flow #2's congestion controller has increased its rate faster than the controller actually expected. Now, flow #1 updates its rate. Its congestion controller detects that the network is not fully saturated and increases its rate. Additionally, the application feeding into flow #1 limits the flow's sending rate to at most 2 Mbit/s.

```
CC_R=7. new_DR=2.
3 a) new_S_CR = 9.33; DELTA = 1.
3 b) FSE_R(f) = 7, DELTA is positive, hence S_CR = 10 + 1 = 11;
     DR = min(2, 7) = 2.
3 c) S_P = 1.5; DR(f) < FSE_R(f), hence TLO = 1/1.5 * 11 - 2 = 5.33.
3 d) new sending rate = min(2, 1/1.5 * 11 + 5.33) = 2.
3 e) FSE_R(f) = 2.
```

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	2	2	2
2	1	0.5	3.33	3.33	3.33

S_CR = 11, TLO = 5.33

Now, the total rate of the two flows is 2 + 3.33 = 5.33 Mbit/s, i.e. the network is significantly underutilized due to the limitation of flow #1. Flow #2 updates its rate. Its congestion controller detects that the network is not fully saturated and increases its rate.

CC_R=4.33. new_DR = infinity.
3 a) new_S_CR = 5.33; DELTA = 1.
3 b) FSE_R(f) = 4.33. DELTA is positive, hence S_CR = 12;
     DR(f) = 4.33.
3 c) S_P = 1.5.
3 d) new sending rate: min(infinity, 0.5/1.5 * 12 + 5.33) = 9.33.
3 e) FSE_R(f) = 9.33, DR(f) = 9.33.

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	2	2	2
2	1	0.5	9.33	9.33	9.33

S_CR = 12, TLO = 0

Now, the total rate of the two flows is 2 + 9.33 = 11.33 Mbit/s. Finally, flow #1 terminates. It sets P to -1 and DR to 0. Let us assume that it terminated late enough for flow #2 to still experience the network in a congested state, i.e. flow #2 decreases its rate in the next iteration.

CC_R = 7.33. new_DR = infinity.
3 a) new_S_CR = 11.33; DELTA = -2.
3 b) FSE_R(f) = 7.33. DELTA is negative, hence S_CR = 9.33;
     DR(f) = 7.33.
3 c) Flow 1 has P = -1, hence it is deleted from the FSE.
     S_P = 0.5.
3 d) new sending rate: min(infinity, 0.5/0.5*9.33 + 0) = 9.33.
3 e) FSE_R(f) = DR(f) = 9.33.

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
2	1	0.5	9.33	9.33	9.33

S_CR = 9.33, TLO = 0

Appendix B. Change log

B.1. Changes from -00 to -01

- Added change log.
- Updated the example algorithm and its operation.

B.2. Changes from -01 to -02

- Included an active version of the algorithm which is simpler.
- Replaced "greedy flow" with "bulk data transfer" and "non-greedy" with "application-limited".
- Updated new_CR to CC_R, and CR to FSE_R for better understanding.

B.3. Changes from -02 to -03

- Included an active conservative version of the algorithm which reduces queue growth and packet loss; added a reference to a technical report that shows these benefits with simulations.
- Moved the passive variant of the algorithm to appendix.

B.4. Changes from -03 to -04

- Extended SBD section.
- Added a note about window-based controllers.

Authors' Addresses

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo,   N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no

Safiqul Islam
University of Oslo
PO Box 1080 Blindern
Oslo,   N-0316
Norway

Phone: +47 22 84 08 37
Email: safiquli@ifi.uio.no

Stein Gjessing
University of Oslo
PO Box 1080 Blindern
Oslo,   N-0316
Norway

Phone: +47 22 85 24 44
Email: steing@ifi.uio.no

Network Working Group
Internet Draft
Intended Status: Informational
Expires: March 16, 2015

X. Zhu, R. Pan
M. A. Ramalho, S. M. de la Cruz
C. Ganzhorn, P. E. Jones
Cisco Systems
S. DAronco
Ecole Polytechnique Federale de Lausanne
September 12, 2014

NADA: A Unified Congestion Control Scheme for Real-Time Media
draft-zhu-rmcat-nada-04

Abstract

This document describes a scheme named network-assisted dynamic adaptation (NADA), a novel congestion control approach 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.

We present here the overall system architecture, recommended behaviors at the sender and the receiver, as well as expected network node operations. Results from extensive simulation studies of the proposed scheme are available upon request.

Status of this Memo

Table of Contents

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 fast 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 losses, queuing delay, and explicit congestion notification (ECN) markings.

Based on the above considerations, we present a scheme named network-assisted dynamic adaptation (NADA). The proposed design benefits from explicit congestion control signals (e.g., ECN markings) from the network, and remains compatible in the presence of implicit signals (delay or loss) only. In addition, it supports weighted bandwidth sharing among competing video flows.

This documentation describes the overall system architecture, recommended designs at the sender and receiver, as well as expected network nodes operations. The signaling mechanism consists of standard RTP timestamp [RFC3550] and standard RTCP feedback reports.

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 RFC 2119 [RFC2119].

3. System Model

The system consists of the following elements:

* Incoming media stream, in the form of consecutive raw video frames and audio samples;

* Media encoder with rate control capabilities. It takes the incoming media stream and encodes it to an RTP stream at a target bit rate R_v. Note that the actual output rate from the encoder R_o may fluctuate randomly around the target R_v. Also, the encoder can only change its rate at rather coarse time intervals, e.g., once every 0.5 seconds.

* RTP sender, responsible for calculating the target bit rate R_n based on network congestion signals (delay or ECN marking reports from the receiver), and for regulating the actual sending rate R_s accordingly. A rate shaping buffer is employed

to absorb the instantaneous difference between video encoder output rate R_v and sending rate R_s. The buffer size L_s, together with R_n, influences the calculation of actual sending rate R_s and video encoder target rate R_v. The RTP sender also generates RTP timestamp in outgoing packets.

* RTP receiver, responsible for measuring and estimating end-to-end delay based on sender RTP timestamp. In the presence of packet losses and ECN markings, it also keeps track of packet loss and ECN marking ratios. It calculates the equivalent delay x_n that accounts for queuing delay, ECN marking, and packet losses, as well as the derivative (i.e., slope of change) of this congestion signal as x'_n. The receiver feeds both information (x_n and x'_n) back to the sender via periodic RTCP reports.

* 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 RED-based ECN marking, and PCN marking using a token bucket algorithm.

In the following, we will elaborate on the respective operations at the network node, the receiver, and the sender.

4. Network Node Operations

We consider three variations of queue management behavior at the network node, leading to either implicit or explicit congestion signals.

4.1 Default behavior of drop tail

In conventional network with drop tail or RED queues, congestion is inferred from the estimation of end-to-end delay and/or packet losses. Packet drops at the queue are detected at the receiver, and contributes to the calculation of the equivalent delay x_n. No special action is required at network node.

4.2 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 [RFC2309]. Calculation of the marking probability involves the following steps:

* upon packet arrival, update smoothed queue size q_avg as:

$$q\_avg = alpha*q + (1-alpha)*q\_avg.$$

The smoothing parameter alpha is a value between 0 and 1. A value of alpha=1 corresponds to performing no smoothing at all.

* calculate marking probability p as:

$$p = 0, \text{ if } q < q\_lo;$$

$$p = p\_max*\frac{q\_avg - q\_lo}{q\_hi - q\_lo}, \text{ if } q\_lo <= q < q\_hi;$$

$$p = 1, \text{ if } q >= q\_hi.$$

Here, q_lo and q_hi corresponds to the low and high thresholds of queue occupancy. The maximum parking probability is p_max.

The ECN markings events will contribute to the calculation of an equivalent delay x_n at the receiver. No changes are required at the sender.

4.3 PCN marking

As a more advanced feature, we also envision network nodes which support PCN marking based on virtual queues. In such a case, the marking probability of the ECN bit in the IP packet header is calculated as follows:

* upon packet arrival, meter packet against token bucket (r,b);

* update token level b_tk;

* calculate the marking probability as:

$$p = 0, \text{ if } b-b\_tk < b\_lo;$$

$$p = p\_max* \frac{b-b\_tk-b\_lo}{b\_hi-b\_lo}, \text{ if } b\_lo<= b-b\_tk <b\_hi;$$

$$p = 1, \text{ if } b-b\_tk>=b\_hi.$$

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 as r=gamma*C, where gamma<1 is the target utilization ratio and C designates link capacity. The maximum marking probability is p_max.

The ECN markings events will contribute to the calculation of an equivalent delay x_n at the receiver. No changes are required at the sender. The virtual queuing mechanism from the PCN marking algorithm will lead to additional benefits such as zero standing queues.

4.4 Comments and Discussions

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

The sender behavior stays the same in the presence of all types of congestion signals: delay, loss, ECN marking due to either RED/ECN or PCN algorithms. This unified approach allows a graceful transition of the scheme as the level of congestion in the network shifts dynamically between different regimes.

5. Receiver Behavior

The receiver periodically monitors end-to-end per-packet statistics in terms of delay, loss, and/or ECN marking ratios. It then aggregates all forms of congestion signals in terms of an equivalent delay, and periodically reports back to the sender.

5.1 Monitoring per-packet statistics

Upon receipt of each packet, the receiver calculates one-way delay as the difference between sender and receiver timestamps:

x_n = t_r,n - t_s,n.

It also maintains its estimate of baseline delay, d_f, as the minimum value of previously observed x_n's over a relatively longer period. This assumes that that sending and receiving clocks are either well-synchronized, or have a relatively stable offset. In our implementation, the baseline delay estimation is updated once every 10 minutes.

Correspondingly, the queuing delay experienced by the packet n is estimated as:

d_n = x_n - d_f.

In addition, the receiver keeps track of both packet loss ratios as p_L via detection of gaps in the packet sequence numbers, and ECN marking ratios as p_M.

5.2 Aggregating congestion signals

The receiver aggregates all three forms of congestion signal in terms of an equivalent delay:

$$x_n = d_n + p_M d_M + p_L d_L, \qquad (1)$$

where d_M is a prescribed fictitious delay value associated with ECN markings (e.g., d_M = 200 ms), and d_L is a prescribed fictitious delay value associated with packet losses (e.g., d_L = 1 second). By introducing a large fictitious delay penalty for ECN marking and packet losses, the proposed scheme leads to low end-to-end actual delays in the presence of such events.

While the value of d_M and d_L are fixed and predetermined in our current design, we also plan to investigate 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).

It should also be noted that in the absence of loss and marking information, the value of x_n falls back to the observed queuing delay d_n for packet n. Our algorithm operates in purely delay-based mode.

5.3 Sending periodic feedback

Periodically, the receiver sends back the most recent value of x_n in RTCP messages, to aid the sender in its calculation of target rate. It also calculates and sends the derivative of x_n as part of the RTCP message:

$$x'_n = \frac{x_n - x_{(n-k)}}{delta}. \qquad (2)$$

Here, the interval between current and previous RTCP messages is denoted as delta, and the corresponding packet indices are n and (n-k), respectively. Typically, the interval between adjacent RTCP receiver reports is on the order of sub-seconds (e.g., 100ms).

The size of acknowledgement packets are typically on the order of tens of bytes, and are significantly smaller than average video packet sizes. Therefore, the bandwidth overhead of the receiver acknowledgement stream is sufficiently low.

5.4 Discussions on delay metrics

Our current design works with relative 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 longer time horizon and subtract that out from the observed absolute OWD value. Such an approach cancels out the fixed clock difference from the sender and receiver clocks, and has been widely adopted by other delay-based congestion control approaches such as LEDBAT [RFC6817]. As discussed in [RFC6817], the time horizon for tracking the minimum OWD needs to be chosen with care: long enough for an opportunity to observe the minimum OWD with zero queuing delay along the path, and sufficiently short so as to timely reflect "true" changes in minimum OWD introduced by route changes and other rare events.

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 account the standing queuing delay as part of the fixed path propagation delay. This will lead to slightly unfair bandwidth sharing amongst the flows.

Alternatively, one could move the per-packet statistical handling to the sender instead, and use RTT in lieu of OWD, assuming that per-packet ACKs are available. The main drawback of this latter approach, on the other hand, is that the scheme will be confused by congestion in the reverse direction.

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

6. Sender Behavior

Figure 1 provides a more detailed view of the NADA sender. Upon receipt of an RTCP report from the receiver, the NADA sender updates its calculation of the reference rate R_n. It further adjusts both the target rate for the live video encoder R_v and the sending rate R_s over the network based on the updated value of R_n, as well as the size of the rate shaping buffer.

```
                    --------------------
                    |                  |
                    |  Reference Rate  | <---- RTCP report
                    |  Calculator      |
                    |                  |
                    --------------------
                            |
                            | R_n
                            |
                ---------------------------
               |                           |
               |                           |
              \ /                         \ /
    --------------------           -----------------
    |                  |           |               |
    |  Video Target    |           | Sending Rate  |
    |  Rate Calculator |           | Calculator    |
    |                  |           |               |
    --------------------           -----------------
       |        /|\                  /|\       |
   R_v|          |                    |        |
       |         -----------------------       |
       |                    |                  | R_s
    ------------            |L_s               |
    |          |            |                  |
    |          |  R_o    --------------       \|/
    |  Encoder |---------->   | | | | |  --------------->
    |          |              | | | | |     video packets
    ------------         --------------
                         Rate Shaping Buffer
```

Figure 1 NADA Sender Structure

The following sections describe these modules in further details, and explain how they interact with each other.

6.1 Video encoder rate control

The video encoder rate control procedure has the following characteristics:

* Rate changes can happen only at large intervals, on the order of seconds.

* Given a target rate R_o, the encoder output rate may randomly fluctuate around it.

* The encoder output rate is further constrained by video content complexity. The range of the final rate output is [R_min, R_max]. Note that it’s content-dependent, and may change over time.

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

6.2 Rate shaping buffer

A rate shaping buffer is employed to absorb any instantaneous mismatch between encoder rate output R_o and regulated sending rate R_s. The size of the buffer evolves from time t-tau to time t as:

```
L_s(t) = max [0, L_s(t-tau)+(R_o-R_s)*tau].
```

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 constrain the size of the shaping buffer. It can either deplete it faster by increasing the sending rate R_s, or limit its growth by reducing the target rate for the video encoder rate control R_v.

6.3 Reference rate calculator

The sender initializes the reference rate R_n as R_min. Upon receipt of a new receiver RTCP reports containing values of x_n and x’_n, it updates the rate as:

```
                 kappa * delta_s
R_n  <-- R_n + ---------------- * (theta-(R_n-R_min)*x_hat) (3)
                    tau_o^2
```

where

```
theta = w*(R_max - R_min)*x_ref.    (4)

x_hat = x_n + eta*tao_o* x’_n       (5)
```

In (3), delta_s refers to the time interval between current and previous rate updates. Note that delta_s is the same as the RTCP report interval at the receiver (see delta from (2)) when the backward path is un-congested.

In (4), R_min and R_max denote the content-dependent rate range the encoder can produce. The weight of priority level is w. The reference congestion signal x_ref is chosen so that the maximum rate of R_max can be achieved when x_hat = w*x_ref.

Proper choice of the scaling parameters eta and kappa in (3) and (5) can ensure system stability so long as the RTT falls below the upper bound of tau_o. In our design, tau_o is chosen as 500ms.

The final target rate R_n is clipped within the range of [R_min, R_max].

Note that the sender does not need any explicit knowledge of the management scheme inside the network. Rather, it reacts to the aggregation of all forms of congestion indications (delay, loss, and marking) via the composite congestion signals x_n and x'_n from the receiver in a coherent manner.

6.4 Video target rate and sending rate calculator

The target rate for the live video encoder is updated based on both the reference rate R_n and the rate shaping buffer size L_s, as follows:

```
                             L_s
        R_v = R_n - beta_v * -------.       (6)
                            tau_v
```

Similarly, the outgoing rate is regulated based on both the reference rate R_n and the rate shaping buffer size L_s, such that:

```
                             L_s
        R_s = R_n + beta_s * -------.       (7)
                            tau_v
```

In (6) and (7), the first term indicates the rate calculated from network congestion feedback alone. The second term indicates 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_n.

Intuitively, the amount of extra rate offset needed to completely drain the rate shaping buffer within the same time frame of encoder rate adaptation tau_v is given by L_s/tau_v. 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 the system from the reference rate point.

6.5 Start-up behavior

The rate adaptation algorithm specified by (3)--(5) naturally leads to a linear rate increase at start-up, when queuing delay stays at zero in the beginning:

```
                       kappa*delta_s
        R_n <-- R_n + ----------------* theta      (8)
                          tau_o^2
```

Given that theta = w*(R_max - R_min)*x_ref, the speed of increase scales with the value of kappa, weight of priority w, and dynamic range of the flow (R_max - R_min).

In practice, one may desire a more aggressive ramp-up behavior during the start-up period, e.g., by doubling the rate upon the receipt of each new RTCP message which reports on near-zero values of x_n and x'_n.

We note here that design of the start-up behavior can be kept orthogonal to the design of the steady-state rate adaptation behavior. This topic is worthy of further investigation separately.

7. Incremental Deployment

One nice property of proposed design is the consistent video end point behavior irrespective of network node variations. This facilitates gradual, incremental adoption of the scheme.

To start off with, the proposed encoder congestion control mechanism can be implemented without any explicit support from the network, and rely solely on observed 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.

8. Implementation Status

The proposed NADA scheme has been implemented in the ns-2 simulation platform [ns2]. Extensive simulation evaluations of an earlier version of the draft are documented in [Zhu-PV13]. Evaluation results of current draft over several test cases in [I-D.draft-sarker-rmcat-eval-test] have been presented at the recent IETF meeting [IETF-90].

The scheme has also been implemented in Linux and has been evaluated in a lab setting also described in [IETF-90]. Evaluation results of NADA in single-flow and multi-flow scenarios from this testbed will be disclosed soon.

9. IANA Considerations

There are no actions for IANA.

10. References

10.1 Normative References

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

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

10.2 Informative References

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC6187] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, December 2012

[ns2] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/

[Zhu-PV13] Zhu, X. and Pan, R., "NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video", in Proc. IEEE International Packet Video Workshop (PV'13). San Jose, CA, USA. December 2013.

[I-D.draft-sarker-rmcat-eval-test] Sarker, Z., Singh, V., Zhu, X., and Ramalho, M., "Test Cases for Evaluating RMCAT Proposals", draft-sarker-rmcat-eval-test-01 (work in progress), June 2014.

[IETF-90] Zhu, X. et al., "NADA Update: Algorithm, Implementa6on, and Test Case Evalua6on Results", presented at IETF 90, https://tools.ietf.org/agenda/90/slides/slides-90-rmcat-6.pdf

Authors' Addresses

Xiaoqing Zhu
Cisco Systems,
12515 Research Blvd.,
Austin, TX 78759, USA
Email: xiaoqzhu@cisco.com

Rong Pan
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: ropan@cisco.com

Michael A. Ramalho
6310 Watercrest Way Unit 203
Lakewood Ranch, FL, 34202, USA
Email: mramalho@cisco.com

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

Charles Ganzhorn
7900 International Drive
International Plaza, Suite 400
Bloomington, MN 55425, USA
Email: cganzhor@cisco.com

Paul E. Jones
7025 Kit Creek Rd.
Research Triangle Park, NC 27709, USA
Email: paulej@packetizer.com

Stefano D’Aronco
EPFL STI IEL LTS4
ELD 220 (Batiment ELD), Station 11
CH-1015 Lausanne, Switzerland
Email: stefano.daronco@epfl.ch

Network Working Group
Internet Draft
Intended Status: Informational
Expires: September 27, 2015

X. Zhu, R. Pan
M. A. Ramalho, S. Mena
C. Ganzhorn, P. E. Jones
Cisco Systems
S. De Aronco
Ecole Polytechnique Federale de Lausanne

March 26, 2015

NADA: A Unified Congestion Control Scheme for Real-Time Media
draft-zhu-rmcat-nada-06

Abstract

Network-Assisted Dynamic Adaptation (NADA) is a novel congestion control scheme for interactive real-time media applications, such as video conferencing. In NADA, the sender regulates its sending rate based on either implicit or explicit congestion signaling in a consistent manner. As one example of explicit signaling, NADA can benefit from explicit congestion notification (ECN) markings from network nodes. It also maintains consistent sender behavior in the absence of explicit signaling by reacting to queuing delay and packet loss.

This document describes the overall system architecture for NADA, as well as recommended behavior at the sender and the receiver.

Status of this Memo

Copyright and License Notice

Table of Contents

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.

Based on the above considerations, this document describes a scheme called network-assisted dynamic adaptation (NADA). The NADA design 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. In addition, it supports weighted bandwidth sharing among competing video flows.

This documentation describes the overall system architecture, recommended designs at the sender and receiver, as well as expected network node operations. The signaling mechanism consists of standard RTP timestamp [RFC3550] and standard RTCP feedback reports.

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 RFC 2119 [RFC2119].

3. System Model

The overall system consists of the following elements:

* Source media stream, in the form of consecutive raw video frames and audio samples;

* Media encoder with rate control capabilities. It takes the source media stream and encodes it to an RTP stream at a target bit rate R_v. Note that the actual output rate from the encoder R_o may fluctuate around the target R_v. Also, the encoder can only change its rate at rather coarse time intervals, e.g., once every 0.5 seconds.

* RTP sender, responsible for calculating the target bit rate R_n 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_v, and for regulating the

actual sending rate R_s accordingly. A rate shaping buffer is employed to absorb the instantaneous difference between video encoder output rate R_v and sending rate R_s. The buffer size L_s, together with R_n, influences the calculation of actual sending rate R_s and video encoder target rate R_v. The RTP sender also generates RTP timestamp in outgoing packets.

* RTP receiver, responsible for measuring and estimating end-to-end delay based on sender RTP timestamp. In the presence of packet loss and ECN markings, it keeps track of packet loss and ECN marking ratios. It calculates the equivalent delay x_n that accounts for queuing delay, ECN marking, and packet loss, as well as the derivative (i.e., rate of change) of this congestion signal as x'_n. The receiver feeds both pieces of information (x_n and x'_n) back to the sender via periodic RTCP reports.

* 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 RED-based ECN marking, and PCN marking using a token bucket algorithm. Note that network node operation is out of scope for the design of NADA.

In the following, we will elaborate on the respective operations at the NADA receiver and sender.

4. NADA Receiver Behavior

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.

4.1 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]. NADA 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.

In mathematical terms, for packet n arriving at the receiver, one-way delay is calculated as:

$$x_n = t_{r,n} - t_{s,n},$$

where t_s,n and t_r,n are sender and receiver timestamps, respectively. A real-world implementation should also properly handle practical issues such as wrap-around in the value of x_n, which are omitted from the above simple expression for brevity.

The base delay, d_f, is estimated as the minimum value of previously observed x_n's over a relatively long period. This assumes that the drift between sending and receiving clocks remains bounded by a small value.

Correspondingly, the queuing delay experienced by the packet n is estimated as:

$$d_n = x_n - d_f.$$

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. We denote the resulting queuing delay value as d_hat_n.

Our current implementation employs a simple 5-point median filter over per-packet queuing delay estimates, followed by an exponential smoothing filter. We have found such relatively simple treatment to suffice 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.

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.

4.2 Estimation of packet loss/marking ratio

The receiver detects packet losses via gaps in the RTP sequence numbers of received packets. It then calculates instantaneous packet loss ratio as the ratio between the number of missing packets over the number of total transmitted packets in the given time window (e.g., during the most recent 500ms). This instantaneous value is passed over an exponential smoothing filter, and the filtered result is reported back to the sender as the observed packet loss ratio p_L.

We note that more sophisticated methods in packet loss ratio calculation, such as that adopted by TFRC [Floyd-CCR00], will likely be beneficial. These alternatives are currently under investigation.

Estimation of packet marking ratio p_M, when ECN is enabled at bottleneck network nodes along the path, will follow the same procedure as above. Here it is assumed that ECN marking information at the IP header are somehow passed along to the transport layer by the receiving endpoint.

4.3 Non-linear warping of delay

In order for a delay-based flow to hold its ground and sustain a reasonable share of bandwidth in the presence of a loss-based flow (e.g., loss-based TCP), it is important to distinguish between different levels of observed queuing delay. For instance, a moderate queuing delay value below 100ms 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.

Inspired by the delay-adaptive congestion window backoff policy in [Budzisz-TON11] -- the work by itself is a window-based congestion control scheme with fair coexistence with TCP -- we devise the following non-linear warping of estimated queuing delay value:

```
        d_tilde_n = (d_hat_n),  if d_hat_n < d_th;

                         (d_max - d_hat_n)^4
        d_tilde_n = d_th --------------------, if  d_th<d_hat_n<d_max;
                          (d_max - d_th)^4

        d_tilde_n = 0,   if d_hat_n > d_max.
```

Here, the queuing delay value is unchanged when it is below the first threshold d_th; it is discounted following a non-linear curve when its value falls between d_th and d_max; above d_max, the high queuing delay value no longer counts toward congestion control.

When queuing delay is in the range (0, d_th), NADA operates in pure delay-based mode if no losses/markings are present. When queuing delay exceeds d_max, NADA reacts to loss/marking only. In between d_th and d_max, the sending rate will converge and stabilize at an operating point with a fairly high queuing delay and non-zero packet loss ratio.

In our current implementation d_th is chosen as 50ms and d_max is chosen as 400ms. The impact of the choice of d_th and d_max will be investigated in future work.

4.4 Aggregating congestion signals

The receiver aggregates all three forms of congestion signal in terms of an equivalent delay:

```
              x_n = d_tilde_n + p_M*d_M + p_L*d_L,      (1)
```

where d_M is a prescribed fictitious delay value associated with ECN markings (e.g., d_M = 200 ms), and d_L is a prescribed fictitious delay value associated with packet losses (e.g., d_L = 1 second). By introducing a large fictitious delay penalty for ECN marking and packet loss, the proposed scheme leads to low end-to-end actual delay in the presence of such events.

While the value of d_M and d_L are fixed and predetermined in the current design, 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) is being studied.

In the absence of ECN marking from the network, the value of x_n falls back to the observed queuing delay d_n for packet n when queuing delay is low and no packets are lost over a lightly congested path. In that case the algorithm operates in purely delay-based mode.

4.5 Estimating receiving rate

Estimation of receiving rate of the flow is fairly straightforward. NADA maintains a recent observation window of 500ms, and simply divides the total size of packets arriving during that window over the time span. The receiving rate is denoted as R_r.

4.6 Sending periodic feedback

Periodically, the receiver feeds back a tuple of the most recent values of <d_hat_n, x_n, x’_n, R_r> in RTCP feedback messages to aid the sender in its calculation of target rate. The queuing delay value d_hat_n is included along with the composite congestion signal x_n so that the sender can decide whether the network is truly underutilized (see Sec. 6.1.1 Accelerated ramp-up).

The value of x’_n corresponds to the derivative (i.e., rate of change) of the composite congestion signal:

```
                x_n - x_(n-k)
        x’_n = ---------------,           (2)
                  delta
```

where the interval between consecutive RTCP feedback messages is denoted as delta. The packet indices corresponding to the current and previous feedback are n and (n-k), respectively.

The choice of target feedback interval needs to strike the right balance between timely feedback and low RTCP feedback message counts. Through simulation studies and frequency-domain analysis, it was determined that a feedback interval below 250ms will not break up the feedback control loop of the NADA congestion control algorithm. Thus, it is recommended to use a target feed interval of 100ms. This will result in a feedback bandwidth of 16Kbps with 200 bytes per feedback message, less than 0.1% overhead for a 1Mbps flow.

4.7 Discussions on 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 LEDBAT [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 queuing delay along the path, and sufficiently short so as to timely reflect "true" changes in minimum OWD introduced by route changes and other rare events.

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 RTT in lieu of OWD, assuming that per-packet ACKs are available. The main drawback of this latter approach is that the scheme will be confused by congestion 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 at the sender. Therefore, comparing the pros and cons regarding which delay metric to adopt can be kept as an orthogonal direction of investigation.

5. NADA Sender Behavior

Figure 1 provides a detailed view of the NADA sender. Upon receipt of an RTCP report from the receiver, the NADA sender updates its calculation of the reference rate R_n. It further adjusts both the target rate for the live video encoder R_v and the sending rate R_s over the network based on the updated value of R_n, as well as the size of the rate shaping buffer.

In the following, we describe these modules in further detail, and explain how they interact with each other.

```
                    --------------------
                    |                  |
                    |  Reference Rate  | <---- RTCP report
                    |  Calculator      |
                    |                  |
                    --------------------
                            |
                            | R_n
                            |
                ---------------------------
               |                          |
               |                          |
              \ /                        \ /
    --------------------           -----------------
    |                  |           |               |
    |  Video Target    |           | Sending Rate  |
    |  Rate Calculator |           | Calculator    |
    |                  |           |               |
    --------------------           -----------------
       |        /|\                    /|\     |
    R_v|         |                      |      |
       |         -----------------------       |
       |                    |                  | R_s
    ------------            |L_s               |
    |          |            |                  |
    |          |  R_o    --------------       \|/
    |  Encoder |---------->   | | | | | ---------------->
    |          |              | | | | |     video packets
    ------------         --------------
                         Rate Shaping Buffer
```

Figure 1 NADA Sender Structure

5.1 Reference rate calculation

The sender initializes the reference rate R_n as R-min by default, or to a value specified by the upper-layer application. [Editor’s note: should proper choice of starting rate value be within the scope of the CC solution?]

The reference rate R_n is calculated based on receiver feedback information regarding queuing delay d_tilde_n, composite congestion signal x_n, its derivative x’_n, as well as the receiving rate R_r. The sender switches between two modes of operation:

* Accelerated ramp up: if the reported queuing delay is close to zero and both values of x_n and x’_n are close to zero, indicating empty queues along the path of the flow and, consequently, underutilized network bandwidth; or

* Gradual rate update: in all other conditions, whereby the receiver reports on a standing or increasing/decreasing queue and/or composite congestion signal.

5.1.1 Accelerated ramp up

In the absence of a non-zero congestion signal to guide the sending rate calculation, the sender needs to ramp up its estimated bandwidth as quickly as possible without introducing excessive queuing delay. Ideally the flow should inflict no more than T_th milliseconds of queuing delay at the bottleneck during the ramp-up process. A typical value of T_th is 50ms.

Note that the sender will be aware of any queuing delay introduced by its rate increase after at least one round-trip time. In addition, the bottleneck bandwidth C is greater than or equal to the receive rate R_r reported from the most recent "no congestion" feedback message. The rate R_n is updated as follows:

```
                               T_th
    gamma = min [gamma_0, ---------------]      (3)
                            RTT_0+delta_0


    R_n  =  (1+gamma) R_r                     (4)
```

In (3) and (4), the multiplier gamma for rate increase is upper-bounded by a fixed ratio gamma_0 (e.g., 20%), as well as a ratio which depends on T_th, base RTT as measured during the non-congested phase, and target ACK interval delta_0. The rationale behind this is that the rate increase multiplier should decrease with the delay in the feedback control loop, and that RTT_0 + delta_0 provides a worst-case estimate of feedback control delay when the network is not congested.

5.1.2. Gradual rate update

When the receiver reports indicate a standing congestion level, NADA operates in gradual update mode, and calculates its reference rate as:

```
                   kappa * delta_s
    R_n  <-- R_n + ---------------- * (theta-(R_n-R_min)*x_hat) (5)
                      tau_o^2
```

where

```
    theta = w*(R_max - R_min)*x_ref.    (6)

    x_hat = x_n + eta*tau_o* x'_n       (7)
```

In (5), delta_s refers to the time interval between current and previous rate updates. Note that delta_s is the same as the RTCP report interval at the receiver (see delta from (2)) when the backward path is un-congested.

In (6), R_min and R_max denote the content-dependent rate range the encoder can produce. The weighting factor reflecting a flow's priority is w. The reference congestion signal x_ref is chosen so that the maximum rate of R_max can be achieved when x_hat = w*x_ref.

Proper choice of the scaling parameters eta and kappa in (5) and (7) can ensure system stability so long as the RTT falls below the upper bound of tau_o. The recommended default value of tau_o is chosen as 500ms.

For both modes of operations, the final reference rate R_n is clipped within the range of [R_min, R_max]. Note also that the sender does not need any explicit knowledge of the management scheme inside the network. Rather, it reacts to the aggregation of all forms of congestion indications (delay, loss, and explicit markings) via the composite congestion signals x_n and x'_n from the receiver in a coherent manner.

5.2 Video encoder rate control

The video encoder rate control procedure has the following characteristics:

* Rate changes can happen only at large intervals, on the order of seconds.

* The encoder output rate may fluctuate around the target rate R_v.

* The encoder output rate is further constrained by video content complexity. The range of the final rate output is [R_min, R_max]. Note that it is content-dependent and may vary over time.

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.

5.3 Rate shaping buffer

A rate shaping buffer is employed to absorb any instantaneous mismatch between encoder rate output R_o and regulated sending rate R_s. The size of the buffer evolves from time t-tau to time t as:

$$L_s(t) = \max [0, L_s(t-\tau)+(R_o-R_s)*\tau].$$

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 constrain the size of the shaping buffer. It can either deplete it faster by increasing the sending rate R_s, or limit its growth by reducing the target rate for the video encoder rate control R_v.

5.4 Adjusting video target rate and sending rate

The target rate for the live video encoder is updated based on both the reference rate R_n and the rate shaping buffer size L_s, as follows:

$$R_v = R_n - \beta_v * \frac{L_s}{\tau_v}. \qquad (8)$$

Similarly, the outgoing rate is regulated based on both the reference rate R_n and the rate shaping buffer size L_s, such that:

$$R_s = R_n + \beta_s * \frac{L_s}{\tau_v}. \qquad (9)$$

In (8) and (9), the first term indicates the rate calculated from network congestion feedback alone. The second term indicates 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_n.

Intuitively, the amount of extra rate offset needed to completely drain the rate shaping buffer within the same time frame of encoder rate adaptation tau_v is given by L_s/tau_v. 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 the system from the reference rate point.

6. 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.

To start off with, the encoder congestion control mechanism can be implemented without any explicit support from the network, and relies solely on observed 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. Implementation Status

The NADA scheme has been implemented in the ns-2 simulation platform [ns2]. Extensive simulation evaluations of an earlier version of the draft are documented in [Zhu-PV13]. Evaluation results of the current draft over several test cases in [I-D.draft-sarker-rmcat-eval-test] have been presented at recent IETF meetings [IETF-90][IETF-91].

The scheme has also been implemented and evaluated in a lab setting as described in [IETF-90]. Preliminary evaluation results of NADA in single-flow and multi-flow scenarios have been presented in [IETF-91].

8. IANA Considerations

There are no actions for IANA.

9. References

9.1 Normative References

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

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

9.2 Informative References

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

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

[Floyd-CCR00] Floyd, S., Handley, M., Padhye, J., and Widmer, J., "Equation-based Congestion Control for Unicast Applications", ACM SIGCOMM Computer Communications Review, vol. 30. no. 4., pp. 43-56, October 2000.

[Budzisz-TON11] Budzisz, L. et al., "On the Fair Coexistence of Loss- and Delay-Based TCP", IEEE/ACM Transactions on Networking, vol. 19, no. 6, pp. 1811-1824, December 2011.

[ns2] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/

[Zhu-PV13] Zhu, X. and Pan, R., "NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video", in Proc. IEEE International Packet Video Workshop (PV’13). San Jose, CA, USA. December 2013.

[I-D.draft-sarker-rmcat-eval-test] Sarker, Z., Singh, V., Zhu, X., and Ramalho, M., "Test Cases for Evaluating RMCAT Proposals", draft-sarker-rmcat-eval-test-01 (work in progress), June 2014.

[IETF-90] Zhu, X. et al., "NADA Update: Algorithm, Implementation, and Test Case Evalua6on Results", presented at IETF 90, https://tools.ietf.org/agenda/90/slides/slides-90-rmcat-6.pdf

[IETF-91] Zhu, X. et al., "NADA Algorithm Update and Test Case Evaluations", presented at IETF 91 Interium, https://datatracker.ietf.org/meeting/91/agenda/rmcat/

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 variations of queue management behavior at the network node, leading to either implicit or explicit congestion signals.

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. Such a simple design ensures that the system can scale easily with a large number of video flows and high link capacity.

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

A.1 Default behavior of drop tail

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 equivalent delay x_n. No special action is required at network node.

A.2 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 [RFC2309]. Calculation of the marking probability involves the following steps:

* upon packet arrival, update smoothed queue size q_{avg} as:

$$q_{avg} = alpha*q + (1-alpha)*q_{avg}.$$

The smoothing parameter alpha is a value between 0 and 1. A value of alpha=1 corresponds to performing no smoothing at all.

* calculate marking probability p as:

$$p = 0, \text{ if } q < q_{lo};$$

$$p = p_{max}*\frac{q_{avg} - q_{lo}}{q_{hi} - q_{lo}}, \text{ if } q_{lo} <= q < q_{hi};$$

$$p = 1, \text{ if } q >= q_{hi}.$$

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

The ECN markings events will contribute to the calculation of an equivalent delay x_n at the receiver. No changes are required at the sender.

A.3 PCN marking

As a more advanced feature, we also envisage network nodes which support PCN marking based on virtual queues. In such a case, the marking probability of the ECN bit in the IP packet header is calculated as follows:

* upon packet arrival, meter packet against token bucket (r,b);

* update token level b_{tk};

* calculate the marking probability as:

$$p = 0, \text{ if } b-b_{tk} < b_{lo};$$

$$p = p_{max} * \frac{b-b_{tk}-b_{lo}}{b_{hi}-b_{lo}}, \text{ if } b_{lo} \le b-b_{tk} < b_{hi};$$

$$p = 1, \text{ if } b-b_{tk} \ge b_{hi}.$$

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 as $r=gamma*C$, where $gamma<1$ is the target utilization ratio and C designates link capacity. The maximum marking probability is p_{max}.

The ECN markings events will contribute to the calculation of an equivalent delay x_n at the receiver. No changes are required at the sender. The virtual queuing mechanism from the PCN marking algorithm will lead to additional benefits such as zero standing queues.

Authors’ Addresses

Xiaoqing Zhu
Cisco Systems,
12515 Research Blvd.,
Austin, TX 78759, USA
Email: xiaoqzhu@cisco.com

Rong Pan
Cisco Systems
510 McCarthy Blvd,
Milpitas, CA 95134, USA
Email: ropan@cisco.com

Michael A. Ramalho
6310 Watercrest Way Unit 203
Lakewood Ranch, FL, 34202, USA
Email: mramalho@cisco.com

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

Charles Ganzhorn
7900 International Drive
International Plaza, Suite 400
Bloomington, MN 55425, USA
Email: charles.ganzhorn@gmail.com

Paul E. Jones
7025 Kit Creek Rd.
Research Triangle Park, NC 27709, USA
Email: paulej@packetizer.com

Stefano D’Aronco
EPFL STI IEL LTS4
ELD 220 (Batiment ELD), Station 11
CH-1015 Lausanne, Switzerland
Email: stefano.daronco@epfl.ch