

Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 31, 2015

S. Holmer
H. Lundin
Google
G. Carlucci
L. De Cicco
S. Mascolo
Politecnico di Bari
June 29, 2015

A Google Congestion Control Algorithm for Real-Time Communication
draft-alvestrand-rmcat-congestion-03

Abstract

This document describes two methods of congestion control when using real-time communications on the World Wide Web (RTCWEB); one delay-based and one loss-based.

It is published as an input document to the RMCAT working group on congestion control for media streams. The mailing list of that working group is rmcat@ietf.org.

Requirements Language

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].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 31, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Mathematical notation conventions	3
2. System model	4
3. Feedback and extensions	5
4. Delay-based control	5
4.1. Arrival-time model	5
4.2. Arrival-time filter	7
4.3. Over-use detector	9
4.4. Rate control	10
4.5. Parameters settings	13
5. Loss-based control	13
6. Interoperability Considerations	15
7. Implementation Experience	15
8. Further Work	15
9. IANA Considerations	16
10. Security Considerations	16
11. Acknowledgements	16
12. References	16
12.1. Normative References	16
12.2. Informative References	17
Appendix A. Change log	17
A.1. Version -00 to -01	17
A.2. Version -01 to -02	17
A.3. Version -02 to -03	18
A.4. rtcweb-03 to rmcat-00	18
A.5. rmcat -00 to -01	18
A.6. rmcat -01 to -02	18
A.7. rmcat -02 to -03	18
Authors' Addresses	19

1. Introduction

Congestion control is a requirement for all applications sharing the Internet resources [RFC2914].

Congestion control for real-time media is challenging for a number of reasons:

- o The media is usually encoded in forms that cannot be quickly changed to accommodate varying bandwidth, and bandwidth requirements can often be changed only in discrete, rather large steps
- o The participants may have certain specific wishes on how to respond - which may not be reducing the bandwidth required by the flow on which congestion is discovered
- o The encodings are usually sensitive to packet loss, while the real-time requirement precludes the repair of packet loss by retransmission

This memo describes two congestion control algorithms that together are able to provide good performance and reasonable bandwidth sharing with other video flows using the same congestion control and with TCP flows that share the same links.

The signaling used consists of experimental RTP header extensions and RTCP messages RFC 3550 [RFC3550] as defined in [abs-send-time], [I-D.alvestrand-rmcat-remb] and [I-D.holmer-rmcat-transport-wide-cc-extensions].

1.1. Mathematical notation conventions

The mathematics of this document have been transcribed from a more formula-friendly format.

The following notational conventions are used:

\bar{X} The variable X , where X is a vector - conventionally marked by a bar on top of the variable name.

\hat{X} An estimate of the true value of variable X - conventionally marked by a circumflex accent on top of the variable name.

$X(i)$ The " i "th value of vector X - conventionally marked by a subscript i .

$[x \ y \ z]$ A row vector consisting of elements x , y and z .

X_{bar}^T The transpose of vector X_{bar} .

$E\{X\}$ The expected value of the stochastic variable X

2. System model

The following elements are in the system:

- o RTP packet - an RTP packet containing media data.
- o Packet group - a set of RTP packets transmitted from the sender uniquely identified by the group departure and group arrival time (absolute send time) [abs-send-time]. These could be video packets, audio packets, or a mix of audio and video packets.
- o Incoming media stream - a stream of frames consisting of RTP packets.
- o RTP sender - sends the RTP stream over the network to the RTP receiver. It generates the RTP timestamp and the abs-send-time header extension
- o RTP receiver - receives the RTP stream, marks the time of arrival.
- o RTCP sender at RTP receiver - sends receiver reports, REMB messages and transport-wide RTCP feedback messages.
- o RTCP receiver at RTP sender - receives receiver reports and REMB messages and transport-wide RTCP feedback messages, reports these to the sender side controller.
- o RTCP receiver at RTP receiver, receives sender reports from the sender.
- o Loss-based controller - takes loss rate measurement, round trip time measurement and REMB messages, and computes a target sending bitrate.
- o Delay-based controller - takes the packet arrival info, either at the RTP receiver, or from the feedback received by the RTP sender, and computes a maximum bitrate which it passes to the loss-based controller.

Together, loss-based controller and delay-based controller implement the congestion control algorithm.

3. Feedback and extensions

There are two ways to implement the proposed algorithm. One where both the controllers are running at the send-side, and one where the delay-based controller runs on the receive-side and the loss-based controller runs on the send-side.

The first version can be realized by using a per-packet feedback protocol as described in [I-D.holmer-rmcat-transport-wide-cc-extensions]. Here, the RTP receiver will record the arrival time and the transport-wide sequence number of each received packet, which will be sent back to the sender periodically using the transport-wide feedback message. The RECOMMENDED feedback interval is once per received video frame or at least once every 30 ms if audio-only or multi-stream. If the feedback overhead needs to be limited this interval can be increased to 100 ms.

The sender will map the received {sequence number, arrival time} pairs to the send-time of each packet covered by the feedback report, and feed those timestamps to the delay-based controller. It will also compute a loss ratio based on the sequence numbers in the feedback message.

The second version can be realized by having a delay-based controller at the receive-side, monitoring and processing the arrival time and size of incoming packets. The sender SHOULD use the abs-send-time RTP header extension [abs-send-time] to enable the receiver to compute the inter-group delay variation. The output from the delay-based controller will be a bitrate, which will be sent back to the sender using the REMB feedback message [I-D.alvestrand-rmcat-remb]. The packet loss ratio is sent back via RTCP receiver reports. At the sender the bitrate in the REMB message and the fraction of packets lost are fed into the loss-based controller, which outputs a final target bitrate. It is RECOMMENDED to send the REMB message as soon as congestion is detected, and otherwise at least once every second.

4. Delay-based control

The delay-based control algorithm can be further decomposed into three parts: an arrival-time filter, an over-use detector, and a rate controller.

4.1. Arrival-time model

This section describes an adaptive filter that continuously updates estimates of network parameters based on the timing of the received packets.

We define the inter-arrival time, $t(i) - t(i-1)$, as the difference in arrival time of two packets or two groups of packets. Correspondingly, the inter-departure time, $T(i) - T(i-1)$, is defined as the difference in departure-time of two packets or two groups of packets. Finally, the inter-group delay variation, $d(i)$, is defined as the difference between the inter-arrival time and the inter-departure time. Or interpreted differently, as the difference between the delay of group i and group $i-1$.

$$d(i) = t(i) - t(i-1) - (T(i) - T(i-1))$$

At the receiving side we are observing groups of incoming packets, where a group of packets is defined as follows:

- o A sequence of packets which are sent within a `burst_time` interval constitute a group. RECOMMENDED value for `burst_time` is 5 ms.
- o In addition, any packet which has an inter-arrival time less than `burst_time` and an inter-group delay variation $d(i)$ less than 0 is also considered being part of the current group of packets. The reasoning behind including these packets in the group is to better handle delay transients, caused by packets being queued up for reasons unrelated to congestion. As an example this has been observed to happen on many Wi-Fi and wireless networks.

An inter-departure time is computed between consecutive groups as $T(i) - T(i-1)$, where $T(i)$ is the departure timestamp of the last packet in the current packet group being processed. Any packets received out of order are ignored by the arrival-time model.

Each group is assigned a receive time $t(i)$, which corresponds to the time at which the last packet of the group was received. A group is delayed relative to its predecessor if $t(i) - t(i-1) > T(i) - T(i-1)$, i.e., if the inter-arrival time is larger than the inter-departure time.

Since the time t_s to send a group of packets of size L over a path with a capacity of C is roughly

$$t_s = L/C$$

we can model the inter-group delay variation as:

$$\begin{aligned}
 d(i) &= L(i)/C(i) - L(i-1)/C(i-1) + w(i) = \\
 &= \frac{L(i)-L(i-1)}{C(i)} + w(i) = dL(i)/C(i) + w(i)
 \end{aligned}$$

Here, $w(i)$ is a sample from a stochastic process W , which is a function of the capacity $C(i)$, the current cross traffic, and the current sent bitrate. C is modeled as being constant as we expect it to vary more slowly than other parameters of this model. We model W as a white Gaussian process. If we are over-using the channel we expect the mean of $w(i)$ to increase, and if a queue on the network path is being emptied, the mean of $w(i)$ will decrease; otherwise the mean of $w(i)$ will be zero.

Breaking out the mean, $m(i)$, from $w(i)$ to make the process zero mean, we get

Equation 1

$$d(i) = dL(i)/C(i) + m(i) + v(i)$$

This is our fundamental model, where we take into account that a large group of packets need more time to traverse the link than a small group, thus arriving with higher relative delay. The noise term represents network jitter and other delay effects not captured by the model.

4.2. Arrival-time filter

The parameters $d(i)$ and $dL(i)$ are readily available for each group of packets, $i > 1$, and we want to estimate $C(i)$ and $m(i)$ and use those estimates to detect whether or not the bottleneck link is over-used. These parameters can be estimated by any adaptive filter - we are using the Kalman filter.

Let

$$\theta_{\text{bar}}(i) = [1/C(i) \quad m(i)]^T$$

and call it the state at time i . We model the state evolution from time i to time $i+1$ as

$$\theta_{\text{bar}}(i+1) = \theta_{\text{bar}}(i) + u_{\text{bar}}(i)$$

where $u_{\text{bar}}(i)$ is the state noise that we model as a stationary process with Gaussian statistic with zero mean and covariance

$$Q(i) = E\{u_{\text{bar}}(i) * u_{\text{bar}}(i)^T\}$$

$Q(i)$ is RECOMMENDED as a diagonal matrix with main diagonal elements as:

$$\text{diag}(Q(i)) = [10^{-13} \ 10^{-3}]^T$$

Given equation 1 we get

$$d(i) = h_{\text{bar}}(i)^T * \theta_{\text{bar}}(i) + v(i)$$

$$h_{\text{bar}}(i) = [dL(i) \ 1]^T$$

where $v(i)$ is zero mean white Gaussian measurement noise with variance $\text{var}_v = \sigma(v,i)^2$

The Kalman filter recursively updates our estimate

$$\theta_{\text{hat}}(i) = [1/C_{\text{hat}}(i) \ m_{\text{hat}}(i)]^T$$

as

$$z(i) = d(i) - h_{\text{bar}}(i)^T * \theta_{\text{hat}}(i-1)$$

$$\theta_{\text{hat}}(i) = \theta_{\text{hat}}(i-1) + z(i) * k_{\text{bar}}(i)$$

$$k_{\text{bar}}(i) = \frac{(E(i-1) + Q(i)) * h_{\text{bar}}(i)}{\text{var}_v_{\text{hat}}(i) + h_{\text{bar}}(i)^T * (E(i-1) + Q(i)) * h_{\text{bar}}(i)}$$

$$E(i) = (I - k_{\text{bar}}(i) * h_{\text{bar}}(i)^T) * (E(i-1) + Q(i))$$

where I is the 2-by-2 identity matrix.

The variance $\text{var}_v(i) = \sigma_v(i)^2$ is estimated using an exponential averaging filter, modified for variable sampling rate

$$\text{var}_v_{\text{hat}}(i) = \max(\beta * \text{var}_v_{\text{hat}}(i-1) + (1-\beta) * z(i)^2, 1)$$

$$\beta = (1-\chi)^{(30/(1000 * f_{\text{max}}))}$$

where $f_{\text{max}} = \max \{1/(T(j) - T(j-1))\}$ for j in $i-K+1, \dots, i$ is the highest rate at which the last K packet groups have been received and χ is a filter coefficient typically chosen as a number in the interval $[0.1, 0.001]$. Since our assumption that $v(i)$ should be zero mean WGN is less accurate in some cases, we have introduced an additional outlier filter around the updates of $\text{var}_v_{\text{hat}}$. If $z(i) > 3*\text{sqrt}(\text{var}_v_{\text{hat}})$ the filter is updated with $3*\text{sqrt}(\text{var}_v_{\text{hat}})$ rather

than $z(i)$. For instance $v(i)$ will not be white in situations where packets are sent at a higher rate than the channel capacity, in which case they will be queued behind each other.

4.3. Over-use detector

The offset estimate $m(i)$, obtained as the output of the arrival-time filter, is compared with a threshold $\gamma_1(i)$. An estimate above the threshold is considered as an indication of over-use. Such an indication is not enough for the detector to signal over-use to the rate control subsystem. A definitive over-use will be signaled only if over-use has been detected for at least γ_2 milliseconds. However, if $m(i) < m(i-1)$, over-use will not be signaled even if all the above conditions are met. Similarly, the opposite state, under-use, is detected when $m(i) < -\gamma_1(i)$. If neither over-use nor under-use is detected, the detector will be in the normal state.

The threshold γ_1 has a remarkable impact on the overall dynamics and performance of the algorithm. In particular, it has been shown that using a static threshold γ_1 , a flow controlled by the proposed algorithm can be starved by a concurrent TCP flow [Pv13]. This starvation can be avoided by increasing the threshold γ_1 to a sufficiently large value.

The reason is that, by using a larger value of γ_1 , a larger queuing delay can be tolerated, whereas with a small γ_1 , the over-use detector quickly reacts to a small increase in the offset estimate $m(i)$ by generating an over-use signal that reduces the delay-based estimate of the available bandwidth $A_{\hat{}}$ (see Section 4.4). Thus, it is necessary to dynamically tune the threshold γ_1 to get good performance in the most common scenarios, such as when competing with loss-based flows.

For this reason, we propose to vary the threshold $\gamma_1(i)$ according to the following dynamic equation:

$$\gamma_1(i) = \gamma_1(i-1) + (t(i)-t(i-1)) * K(i) * (|m(i)|-\gamma_1(i-1))$$

with $K(i)=K_d$ if $|m(i)| < \gamma_1(i-1)$ or $K(i)=K_u$ otherwise. The rationale is to increase $\gamma_1(i)$ when $m(i)$ is outside of the range $[-\gamma_1(i-1), \gamma_1(i-1)]$, whereas, when the offset estimate $m(i)$ falls back into the range, γ_1 is decreased. In this way when $m(i)$ increases, for instance due to a TCP flow entering the same bottleneck, $\gamma_1(i)$ increases and avoids the uncontrolled generation of over-use signals which may lead to starvation of the flow controlled by the proposed algorithm [Pv13]. Moreover, $\gamma_1(i)$ SHOULD NOT be updated if this condition holds:

$$|m(i)| - \gamma_1(i) > 15$$

It is also RECOMMENDED to clamp $\gamma_1(i)$ to the range $[6, 600]$, since a too small $\gamma_1(i)$ can cause the detector to become overly sensitive.

On the other hand, when $m(i)$ falls back into the range $[-\gamma_1(i-1), \gamma_1(i-1)]$ the threshold $\gamma_1(i)$ is decreased so that a lower queuing delay can be achieved.

It is RECOMMENDED to choose $K_u > K_d$ so that the rate at which γ_1 is increased is higher than the rate at which it is decreased. With this setting it is possible to increase the threshold in the case of a concurrent TCP flow and prevent starvation as well as enforcing intra-protocol fairness. RECOMMENDED values for $\gamma_1(0)$, γ_2 , K_u and K_d are respectively 12.5 ms, 10 ms, 0.01 and 0.00018.

4.4. Rate control

The rate control is split in two parts, one controlling the bandwidth estimate based on delay, and one controlling the bandwidth estimate based on loss. Both are designed to increase the estimate of the available bandwidth A_{hat} as long as there is no detected congestion and to ensure that we will eventually match the available bandwidth of the channel and detect an over-use.

As soon as over-use has been detected, the available bandwidth estimated by the delay-based controller is decreased. In this way we get a recursive and adaptive estimate of the available bandwidth.

In this document we make the assumption that the rate control subsystem is executed periodically and that this period is constant.

The rate control subsystem has 3 states: Increase, Decrease and Hold. "Increase" is the state when no congestion is detected; "Decrease" is the state where congestion is detected, and "Hold" is a state that waits until built-up queues have drained before going to "increase" state.

The state transitions (with blank fields meaning "remain in state") are:

Signal \ State	Hold	Increase	Decrease
Over-use	Decrease	Decrease	
Normal	Increase		Hold
Under-use		Hold	Hold

The subsystem starts in the increase state, where it will stay until over-use or under-use has been detected by the detector subsystem. On every update the delay-based estimate of the available bandwidth is increased, either multiplicatively or additively, depending on its current state.

The system does a multiplicative increase if the current bandwidth estimate appears to be far from convergence, while it does an additive increase if it appears to be closer to convergence. We assume that we are close to convergence if the currently incoming bitrate, $R_{\text{hat}}(i)$, is close to an average of the incoming bitrates at the time when we previously have been in the Decrease state. "Close" is defined as three standard deviations around this average. It is RECOMMENDED to measure this average and standard deviation with an exponential moving average with the smoothing factor 0.95, as it is expected that this average covers multiple occasions at which we are in the Decrease state. Whenever valid estimates of these statistics are not available, we assume that we have not yet come close to convergence and therefore remain in the multiplicative increase state.

If $R_{\text{hat}}(i)$ increases above three standard deviations of the average max bitrate, we assume that the current congestion level has changed, at which point we reset the average max bitrate and go back to the multiplicative increase state.

$R_{\text{hat}}(i)$ is the incoming bitrate measured by the delay-based controller over a T seconds window:

$$R_{\text{hat}}(i) = 1/T * \sum(L(j)) \text{ for } j \text{ from } 1 \text{ to } N(i)$$

$N(i)$ is the number of packets received the past T seconds and $L(j)$ is the payload size of packet j. A window between 0.5 and 1 second is RECOMMENDED.

During multiplicative increase, the estimate is increased by at most 8% per second.

```
eta = 1.08^min(time_since_last_update_ms / 1000, 1.0)
A_hat(i) = eta * A_hat(i-1)
```

During the additive increase the estimate is increased with at most half a packet per response_time interval. The response_time interval is estimated as the round-trip time plus 100 ms as an estimate of over-use estimator and detector reaction time.

```
response_time_ms = 100 + rtt_ms
beta = 0.5 * min(time_since_last_update_ms / response_time_ms, 1.0)
A_hat(i) = A_hat(i-1) + max(1000, beta * expected_packet_size_bits)
```

expected_packet_size_bits is used to get a slightly slower slope for the additive increase at lower bitrates. It can for instance be computed from the current bitrate by assuming a frame rate of 30 frames per second:

```
bits_per_frame = A_hat(i-1) / 30
packets_per_frame = ceil(bits_per_frame / (1200 * 8))
avg_packet_size_bits = bits_per_frame / packets_per_frame
```

Since the system depends on over-using the channel to verify the current available bandwidth estimate, we must make sure that our estimate does not diverge from the rate at which the sender is actually sending. Thus, if the sender is unable to produce a bit stream with the bitrate the congestion controller is asking for, the available bandwidth estimate should stay within a given bound. Therefore we introduce a threshold

```
A_hat(i) < 1.5 * R_hat(i)
```

When an over-use is detected the system transitions to the decrease state, where the delay-based available bandwidth estimate is decreased to a factor times the currently incoming bitrate.

```
A_hat(i) = alpha * R_hat(i)
```

alpha is typically chosen to be in the interval [0.8, 0.95], 0.85 is the RECOMMENDED value.

When the detector signals under-use to the rate control subsystem, we know that queues in the network path are being emptied, indicating that our available bandwidth estimate A_hat is lower than the actual available bandwidth. Upon that signal the rate control subsystem will enter the hold state, where the receive-side available bandwidth

estimate will be held constant while waiting for the queues to stabilize at a lower level - a way of keeping the delay as low as possible. This decrease of delay is wanted, and expected, immediately after the estimate has been reduced due to over-use, but can also happen if the cross traffic over some links is reduced.

It is RECOMMENDED that the routine to update $A_{\text{hat}}(i)$ is run at least once every `response_time` interval.

4.5. Parameters settings

Parameter	Description	RECOMMENDED Value
<code>burst_time</code>	Time limit in milliseconds between packet bursts which identifies a group	5 ms
<code>Q</code>	State noise covariance matrix	$\text{diag}(Q(i)) = [10^{-13} \ 10^{-3}]^T$
<code>E(0)</code>	Initial value of the system error covariance	$\text{diag}(E(0)) = [100 \ 0.1]^T$
<code>chi</code>	Coefficient used for the measured noise variance	[0.1, 0.001]
<code>gamma_1(0)</code>	Initial value for the adaptive threshold	12.5 ms
<code>gamma_2</code>	Time required to trigger an overuse signal	10 ms
<code>K_u</code>	Coefficient for the adaptive threshold	0.01
<code>K_d</code>	Coefficient for the adaptive threshold	0.00018
<code>T</code>	Time window for measuring the received bitrate	[0.5, 1] s
<code>alpha</code>	Decrease rate factor	0.85

Table 1: RECOMMENDED values for delay based controller

Table 1

5. Loss-based control

A second part of the congestion controller bases its decisions on the round-trip time, packet loss and available bandwidth estimates A_{hat} received from the delay-based controller. The available bandwidth

estimates computed by the loss-based controller are denoted with As_hat .

The available bandwidth estimates A_hat produced by the delay-based controller are only reliable when the size of the queues along the path sufficiently large. If the queues are very short, over-use will only be visible through packet losses, which are not used by the delay-based controller.

The loss-based controller SHOULD run every time feedback from the receiver is received.

- o If 2-10% of the packets have been lost since the previous report from the receiver, the sender available bandwidth estimate $As_hat(i)$ will be kept unchanged.
- o If more than 10% of the packets have been lost a new estimate is calculated as $As_hat(i) = As_hat(i-1)(1-0.5p)$, where p is the loss ratio.
- o As long as less than 2% of the packets have been lost $As_hat(i)$ will be increased as $As_hat(i) = 1.05(As_hat(i-1))$

The new bandwidth estimate is lower-bounded by the TCP Friendly Rate Control formula [RFC3448] and upper-bounded by the delay-based estimate of the available bandwidth $A_hat(i)$, where the delay-based estimate has precedence:

$$As_hat(i) \geq \frac{8s}{R\sqrt{2bp/3} + (t_RTO*(3\sqrt{3bp/8}*p*(1+32p^2)))}$$

$$As_hat(i) \leq A_hat(i)$$

where b is the number of packets acknowledged by a single TCP acknowledgment (set to 1 per TFRC recommendations), t_RTO is the TCP retransmission timeout value in seconds (set to $4*R$) and s is the average packet size in bytes. R is the round-trip time in seconds.

(The multiplication by 8 comes because TFRC is computing bandwidth in bytes, while this document computes bandwidth in bits.)

In words: The loss-based estimate will never be larger than the delay-based estimate, and will never be lower than the estimate from the TFRC formula except if the delay-based estimate is lower than the TFRC estimate.

We motivate the packet loss thresholds by noting that if the transmission channel has a small amount of packet loss due to over-use, that amount will soon increase if the sender does not adjust his bitrate. Therefore we will soon enough reach above the 10% threshold and adjust $As_hat(i)$. However, if the packet loss ratio does not increase, the losses are probably not related to self-inflicted congestion and therefore we should not react on them.

6. Interoperability Considerations

In case a sender implementing these algorithms talks to a receiver which do not implement any of the proposed RTCP messages and RTP header extensions, it is suggested that the sender monitors RTCP receiver reports and uses the fraction of lost packets and the round-trip time as input to the loss-based controller. The delay-based controller should be left disabled.

7. Implementation Experience

This algorithm has been implemented in the open-source WebRTC project, has been in use in Chrome since M23, and is being used by Google Hangouts.

Deployment of the algorithm have revealed problems related to, e.g, congested or otherwise problematic WiFi networks, which have led to algorithm improvements. The algorithm has also been tested in a multi-party conference scenario with a conference server which terminates the congestion control between endpoints. This ensures that no assumptions are being made by the congestion control about maximum send and receive bitrates, etc., which typically is out of control for a conference server.

8. Further Work

This draft is offered as input to the congestion control discussion.

Work that can be done on this basis includes:

- o Considerations of integrated loss control: How loss and delay control can be better integrated, and the loss control improved.
- o Considerations of locus of control: evaluate the performance of having all congestion control logic at the sender, compared to splitting logic between sender and receiver.
- o Considerations of utilizing ECN as a signal for congestion estimation and link over-use detection.

9. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

10. Security Considerations

An attacker with the ability to insert or remove messages on the connection would have the ability to disrupt rate control. This could make the algorithm to produce either a sending rate under-utilizing the bottleneck link capacity, or a too high sending rate causing network congestion.

In this case, the control information is carried inside RTP, and can be protected against modification or message insertion using SRTP, just as for the media. Given that timestamps are carried in the RTP header, which is not encrypted, this is not protected against disclosure, but it seems hard to mount an attack based on timing information only.

11. Acknowledgements

Thanks to Randell Jesup, Magnus Westerlund, Varun Singh, Tim Panton, Soo-Hyun Choo, Jim Gettys, Ingemar Johansson, Michael Welzl and others for providing valuable feedback on earlier versions of this draft.

12. References

12.1. Normative References

- [I-D.alvestrand-rmcat-remb]
Alvestrand, H., "RTCP message for Receiver Estimated Maximum Bitrate", draft-alvestrand-rmcat-remb-03 (work in progress), October 2013.
- [I-D.holmer-rmcat-transport-wide-cc-extensions]
Holmer, S., Flodman, M., and E. Sprang, "RTP Extensions for Transport-wide Congestion Control", draft-holmer-rmcat-transport-wide-cc-extensions-00 (work in progress), March 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC3448] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [abs-send-time]
"RTP Header Extension for Absolute Sender Time",
<<http://www.webrtc.org/experiments/rtp-hdext/abs-send-time>>.

12.2. Informative References

- [Pv13] De Cicco, L., Carlucci, G., and S. Mascolo, "Understanding the Dynamic Behaviour of the Google Congestion Control", Packet Video Workshop , December 2013.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

Appendix A. Change log

A.1. Version -00 to -01

- o Added change log
- o Added appendix outlining new extensions
- o Added a section on when to send feedback to the end of section 3.3 "Rate control", and defined min/max FB intervals.
- o Added size of over-bandwidth estimate usage to "further work" section.
- o Added startup considerations to "further work" section.
- o Added sender-delay considerations to "further work" section.
- o Filled in acknowledgments section from mailing list discussion.

A.2. Version -01 to -02

- o Defined the term "frame", incorporating the transmission time offset into its definition, and removed references to "video frame".

- o Referred to "m(i)" from the text to make the derivation clearer.
- o Made it clearer that we modify our estimates of available bandwidth, and not the true available bandwidth.
- o Removed the appendixes outlining new extensions, added pointers to REMB draft and RFC 5450.

A.3. Version -02 to -03

- o Added a section on how to process multiple streams in a single estimator using RTP timestamps to NTP time conversion.
- o Stated in introduction that the draft is aimed at the RMCAT working group.

A.4. rtcweb-03 to rmcatt-00

Renamed draft to link the draft name to the RMCAT WG.

A.5. rmcatt -00 to -01

Spellcheck. Otherwise no changes, this is a "keepalive" release.

A.6. rmcatt -01 to -02

- o Added Luca De Cicco and Saverio Mascolo as authors.
- o Extended the "Over-use detector" section with new technical details on how to dynamically tune the offset `gamma_1` for improved fairness properties.
- o Added reference to a paper analyzing the behavior of the proposed algorithm.

A.7. rmcatt -02 to -03

- o Swapped receiver-side/sender-side controller with delay-based/loss-based controller as there is no longer a requirement to run the delay-based controller on the receiver-side.
- o Removed the discussion about multiple streams and transmission time offsets.
- o Introduced a new section about "Feedback and extensions".
- o Improvements to the threshold adaptation in the "Over-use detector" section.

- o Swapped the previous MIMD rate control algorithm for a new AIMD rate control algorithm.

Authors' Addresses

Stefan Holmer
Google
Kungsbron 2
Stockholm 11122
Sweden

Email: holmer@google.com

Henrik Lundin
Google
Kungsbron 2
Stockholm 11122
Sweden

Gaetano Carlucci
Politecnico di Bari
Via Orabona, 4
Bari 70125
Italy

Email: gaetano.carlucci@poliba.it

Luca De Cicco
Politecnico di Bari
Via Orabona, 4
Bari 70125
Italy

Email: l.decicco@poliba.it

Saverio Mascolo
Politecnico di Bari
Via Orabona, 4
Bari 70125
Italy

Email: mascolo@poliba.it

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

R. Jesup
Mozilla
Feb 25, 2013

Congestion Control Requirements For RMCAT
draft-jesup-rmcat-reqs-01

Abstract

Congestion control is needed for all data transported across the Internet, in order to promote fair usage and prevent congestion collapse. The requirements for interactive, point-to-point real time multimedia, which needs by low-delay, semi-reliable data delivery, are different from the requirements for bulk transfer like FTP or bursty transfers like Web pages, and the TCP algorithms are not suitable for this traffic.

This document attempts to describe a set of requirements that can be used to evaluate other congestion control mechanisms in order to figure out their fitness for this purpose, and in particular to provide a set of possible requirements for proposals coming out of the RMCAT Working Group.

This document is derived from draft-jesup-rtp-congestion-reqs [I-D.jesup-rtp-congestion-reqs].

Requirements Language

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].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
2. Requirements	4
3. IANA Considerations	7
4. Security Considerations	8
5. Acknowledgements	8
6. References	8
6.1. Normative References	8
6.2. Informative References	8
Author's Address	9

1. Introduction

The traditional TCP congestion control requirements were developed in order to promote efficient use of the Internet for reliable bulk transfer of non-time-critical data, such as transfer of large files. They have also been used successfully to govern the reliable transfer of smaller chunks of data in "as fast as possible" mode, such as when fetching Web pages.

These algorithms have also been used for transfer of media streams that are viewed in a non-interactive manner, such as "streaming" video, where having the data ready when the viewer wants it is important, but the exact timing of the delivery is not.

When doing real time interactive media, the requirements are different; one needs to provide the data continuously, within a very limited time window (no more than 100s of milliseconds end-to-end delay), the sources of data may be able to adapt the amount of data that needs sending within fairly wide margins, and may tolerate some amount of packet loss, but since the data is generated in real time, sending "future" data is impossible, and since it's consumed in real time, data delivered late is useless.

One particular protocol portfolio being developed for this use case is WebRTC [I-D.ietf-rtcweb-overview], where one envisions sending multiple RTP-based flows between two peers, in conjunction with data flows, all at the same time, without having special arrangements with the intervening service providers.

Given that this use case is the focus of this document, use cases involving noninteractive media such as YouTube-like video streaming, and use cases using multicast/broadcast-type technologies, are out of scope.

The terminology defined in [I-D.ietf-rtcweb-overview] is used in this memo.

2. Requirements

1. The congestion control algorithm must attempt to provide as-low-as-possible-delay transit for real-time traffic while still providing a useful amount of bandwidth, even when faced with intermediate bottlenecks and competing flows. There may be lower limits on the amount of bandwidth that is useful, but this is largely application-specific and the application may be able to modify or remove flows in order allow some useful flows to get enough bandwidth. (Example: not enough bandwidth for low-

latency video+audio, but enough for audio-only.)

- A. It should also deal well with routing changes and interface changes (WiFi to 3G data, etc) which may radically change the bandwidth available.
- 2. The algorithm must be fair to other flows, both realtime flows (such as other instances of itself), and TCP flows, both long-lived and bursts such as the traffic generated by a typical web browsing session. Note that 'fair' is a rather hard-to-define term.
 - A. The algorithm must not overreact to short-term bursts (such as web-browsing) which can quickly saturate a local-bottleneck router or link, but also clear quickly, and should recover quickly when the burst ends.
 - B. We will need make some evaluation of fairness, but deciding what is "fair" is a tough question and likely to be partially subjective, but we should specify some of the inputs needed in order to select among algorithms and tunings presented as options.
- 3. The algorithm should where possible merge information across multiple RTP streams between the same endpoints, whether or not they're multiplexed on the same ports, in order to allow congestion control of the set of streams together instead of as multiple independent streams. This allows better overall bandwidth management, faster response to changing conditions, and fairer sharing of bandwidth with other network users.
 - A. If possible, it should also share information and adaptation with other non-RTP flows between the same endpoints, such as a WebRTC data channel
- 4. The algorithm should not require any special support from network elements (ECN, etc). As much as possible, it should leverage existing information about the incoming flows to provide feedback to the sender. Examples of this information are the packet arrival times, acknowledgments and feedback, packet timestamps, packet sizes, packet losses. Extra information could be added to the packets to provide more detailed information on actual send times (as opposed to sampling times), but should not be required.
 - A. When additional input signals such as ECN are available, they should be utilized if possible.

5. Since the assumption here is a set of RTP streams, the backchannel typically should be done via RTCP; the alternative would be to include it in a reverse RTP channel using header extensions.
 - A. In order to react sufficiently quickly, the AVPF/SAVPF RTP profile[RFC4585] must be used
 - B. Note that in some cases, backchannel messages may be delayed until the RTCP channel can be allocated enough bandwidth, even under AVPF rules. This may also imply negotiating a higher maximum percentage for RTCP data or allowing RMCAT solutions to violate or modify the rules specified for AVPF.
 - C. Note that RTCP is of course unreliable
 - D. Bandwidth for the feedback messages should be minimized (such as via RFC 5506 [RFC5506]) to allow RTCP without SR/RR)
 - E. Header extensions would avoid the RTCP timing rules issues, and allow the application to allocate bandwidth as needed for the congestion algorithm.
 - F. Backchannel data should be minimized to avoid taking too much reverse-channel bandwidth (since this will often be used in a bidirectional set of flows). In areas of stability, backchannel data may be sent more infrequently so long as algorithm stability and fairness are maintained. When the channel is unstable or has not yet reached equilibrium after a change, backchannel feedback may be more frequent and use more reverse-channel bandwidth. This is an area with considerable flexibility of design, and different approaches to backchannel messages and frequency are expected to be evaluated.
6. Where possible and helpful, the algorithm should leverage and piggyback on other RTP/RTCP communications, such as SR/RR, rctp-fb PLI, RPSI, SLI or application-specific NACK messages (such as for loss information), and also reverse-direction RTP.
7. The algorithm should sense the unexpected lack of backchannel information as a possible indication of a channel overuse problem and react accordingly to avoid burst events causing a congestion collapse.
8. It should attempt to avoid bandwidth 'collapse' when facing a long-lived saturating TCP flow or flows. (I.e. a classic delay-sensitive algorithm will reduce bandwidth to keep delay down)

until the TCP flow has all the bandwidth). See the Cx-TCP algorithm discussed in a recent Transactions On Networking [cx-tcp] for an example of a delay-sensitive congestion-control algorithm that transitions to a loss-based mode when competing with TCP flows - at the cost of increased delay.

9. The algorithm should be stable and low-delay when faced with active queue management (AQM) such as RED [RFC2309] or CoDel [I-D.nichols-tsvwg-codel] in the channel.
10. The algorithm should quickly adapt to initial network conditions at the start of a flow. This should occur both if the initial bandwidth is above or below the bottleneck bandwidth.
 - A. The startup adaptation may be faster than adaptation later in a flow. It should allow for both slow-start operation (adapt up) and history-based startup (start at a point expected to be at or below channel bandwidth from historical information, which may need to adapt down quickly if the initial guess is wrong). Starting too low and/or adapting up too slowly can cause a critical point in a personal communication to be poor ("Hello!").
 - B. Starting over-bandwidth causes other problems for user experience, so there's a tension here.
 - C. Alternative methods to help startup like probing during setup with dummy data may be useful in some applications.
11. It should be evaluated in how it works both with backbone-router bottlenecks, (asymmetric) local-loop bottlenecks, and local-lan (WiFi/etc) bottlenecks, and in competition with varying numbers and types of streams (TCP, TCP variants in use, LEDBAT [I-D.ietf-ledbat-congestion], inflexible VoIP UDP flows).
12. It should be stable if the RTP streams are halted or discontinuous (VAD/DTX).
 - A. After a resumption of RTP data it may adapt more quickly (similar to the start of a flow), and previous bandwidth estimates may need to be aged or thrown away.

3. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an

RFC.

4. Security Considerations

An attacker with the ability to delete, delay or insert messages in the flow can fake congestion signals, unless they are passed on a tamper-proof path. Since some possible algorithms depend on the timing of packet arrival, even a traditional protected channel does not fully mitigate such attacks.

An attack that reduces bandwidth is not necessarily significant, since an on-path attacker could break the connection by discarding all packets. Attacks that increase the perceived available bandwidth are conceivable, and need to be evaluated.

Algorithm designers SHOULD consider the possibility of malicious on-path attackers.

5. Acknowledgements

This document is the result of discussions in various fora of the WebRTC effort, in particular on the `rtp-congestion@alvestrand.no` mailing list. Many people contributed their thoughts to this.

6. References

6.1. Normative References

- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.

6.2. Informative References

- [I-D.ietf-ledbat-congestion]
Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind,

"Low Extra Delay Background Transport (LEDBAT)",
draft-ietf-ledbat-congestion-10 (work in progress),
September 2012.

[I-D.jesup-rtp-congestion-reqs]

Jesup, R. and H. Alvestrand, "Congestion Control
Requirements For Real Time Media",
draft-jesup-rtp-congestion-reqs-00 (work in progress),
March 2012.

[I-D.nichols-tsvwg-codel]

Nichols, K., "Controlled Delay Active Queue Management",
draft-nichols-tsvwg-codel-00 (work in progress),
July 2012.

[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.

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

[cx-tcp] Budzisz, L., Stanojevic, R., Schlote, A., Baker, F., and
R. Shorten, "On the Fair Coexistence of Loss- and Delay-
Based TCP", December 2011.

Author's Address

Randell Jesup
Mozilla
USA

Email: randell-ietf@jesup.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 26, 2013

P. O'Hanlon
University of Oxford
K. Carlberg
G11
April 24, 2013

Congestion control algorithm for lower latency and lower loss media
transport
draft-ohanlon-rmcat-dflow-02

Abstract

This memo provides a design for a congestion control algorithm, for media transport, which aims to provide for lower delay and lower loss communications.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 26, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Conventions, Definitions and Acronyms	3
3. Background	3
3.1. TFRC	3
3.2. Delay-Based schemes	4
4. Objectives	5
5. Design Outline	6
5.1. Delay Composition	6
5.2. Delay Measurement	6
5.3. Congestion Detection	7
5.4. Slow Start	7
5.5. Loss-mode	7
6. Further Work	8
7. IANA Considerations	8
8. Security Considerations	8
9. References	8
9.1. Normative References	8
9.2. Informative References	8
Authors' Addresses	9

1. Introduction

This memo outlines DFlow, a congestion control algorithm that aims to minimise delay and loss by using delay-based techniques. The scheme is based upon TCP Friendly Rate Control (TFRC) [RFC5348], and adds a delay-based congestion detection scheme which feeds into a 'congestion event history' mechanism based upon TFRC's loss history. This then provides for a 'congestion event rate' which drives the TCP equation.

Congestion control that aims to minimise the delay is important for real-time streams as high delay can render the communication unacceptable [ITU.G114.2003]. On today's Internet a number of paths have an excess of buffering which can lead to persistent high latencies, which has become known as the Bufferbloat phenomenon. These problems are particularly apparent with loss-based congestion control schemes such as TCP, as they operate by filling the queues on a path till loss occurs, thus maximising the delay. The unfortunate consequence is that loss-based approaches not only lead to high delay for their own packets but also introduce delays and losses for all other flows that traverse those same filled queues.

Thus when competing with TCP, without the widespread deployment of Active Queue Management that aims to minimise delay, (e.g. Codel [I-D.nichols-tsvwg-codel]), it is not possible to maintain low delay

as TCP will do its best to keep the queues full and maximise the delay.

However there are many paths where the flows are not competing directly with TCP and where delay may be minimised.

The DFlow scheme can transport media with low delay and loss on paths where there is no direct competition with TCP in the same queue. Though we are currently testing some techniques to enable it compete with loss-based schemes (at the expense of delay) but they will be included in a later version of the draft. In simulations it has been seen to be reasonably fair when competing with other DFlow streams.

2. Conventions, Definitions and Acronyms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Background

Whilst the existing standard for media transport, Real-time Transport Protocol (RTP) [RFC3550], suggests that congestion control should be employed, in practice many systems tend to use fixed or variable bit rate UDP and do very little or no adaptation to their network environment. Most of the existing work on real-time congestion control algorithms has been rooted in TCP-friendly approaches but with smoother adaptation cycles. TCP congestion control is unsuitable for interactive media for a number of reasons including the fact that it is loss-based so it maximises the latency on a path, it changes its transmit rate to quickly for multimedia, and favours reliability over timeliness. Various TCP-friendly congestion control algorithms such as TFRC [RFC5348], Sisalem's LDA+ [SisalemLDA.2000], and Choi's TCP Friendly Window Control (TFWC) [ChoiTFWC.2007] have been devised for media transport, that attempt to smooth the short-term variation in sending rate. More recently there have been development of some delay-based schemes which aim to provide for low delay.

3.1. TFRC

TFRC is a rate based receiver driven congestion control algorithm which utilises the Padhye TCP equation to provide a smoothed TCP-friendly rate. The sender explicitly sets the transmission rate, using the TCP equation driven by the loss event rate which is measured and fed back by the receiver, where a loss event consists of one or more packet losses within a single RTT. It utilises a

weighted smoothed loss event rate, and EWMA smoothed RTT, as input to the TCP equation which enables it to achieve a smoother rate adaptation that provides for a more suitable transport for multimedia. TFRC was primarily aimed at streaming media delivery where a smooth rate and TCP-friendliness are more important than low latency operation.

However there are number of issues with TFRC as regards real-time media transport:

Loss-based operation: Firstly since it is a loss-based based scheme the latency is maximised which is a problem for real-time transport over heavily buffered paths. The other problem with loss-based protocols is that they rely on a certain level of packet loss which can be an issue for media traffic since lost media packet cannot usually be retransmitted in time. This problem becomes more of a concern at lower transmission rates since the TCP equation requires a corresponding increase in loss rates.

Bursty media flows: Many media flows exhibit bursty behaviour due to a number of factors. Firstly there may be negative bursts (i.e. gaps) due to silence or low motion which can lead oscillatory behaviours due to the data-limited and/or idle behaviours. Secondly there may be positive bursts (i.e. larger than normal) can also be due to the bursty nature of the media and codec (e.g. I-frames) which can be lead to drops or increased latency. Whilst the current version of TFRC [RFC5348] has attempted to address some of these issues, they are still a concern.

Small RTT environments: When operating in low RTT environments (<5ms), such as a LAN, systems implementing TFRC can have problems with scheduling packet transmissions as inter-packet timings can be lower than application level clock granularity. Whilst the current version of TFRC [RFC5348] has attempted to address these issues, they can still be a concern in some low RTT environments.

Variable packet sizes: As originally designed TFRC will only operate correctly when packet sizes are close to MTU size, and when the packet sizes are much smaller fairness issues arise. Although there have been attempts to address this problem for small packets [RFC4828], it is not clear how to deal with flows that do vary their packet sizes substantially. However this issue is only really a marked problem with lower bit rate video flows or variable packet rate audio.

3.2. Delay-Based schemes

In the last few years there has been a renewed interest in the use of delay based congestion control for media, with a slightly different emphasis to that of the history of TCP based approaches such as Jain's CARD, Wang and Crowcroft's Tri-S, Brakmo's Vegas, Tan et al's Compound TCP, and more recently Budzisz's CxTCP [BudziszCxTCP.2011]. Where the primary goal with media based transports is to actually minimise the latency of the flow, as opposed to just using delay as an early indication of loss. This is of particular relevance on paths with large queues, as is the case with a number of today's Internet paths. In 2007 Ghanbari et al [GhanbariFuzzy.2007] did some pioneering work on delay-based video congestion control using fuzzy logic based systems. Recently there has been on going activity in the IETF as part of the Low Extra Delay Background Transport (LEDBAT) Working Group which aims to provide a less than best effort delay-based transport with lower delay. However [RFC6817] specifies a one-way queuing delay target of 100ms which is quite a high baseline for interactive media, considering the recommended total one-way delay limit for a VoIP call should be less than 150ms [ITU.G114.2003].

4. Objectives

The objectives of DFlow are to provide for low delay and low loss media transport when possible. We also aim to provide (in a future version of the draft) mechanisms to provide for better burst management, and loss-mode operation.

Lower Delay: The one-way delay should be kept well within the acceptable levels of 150ms, and MUST NOT exceed 400ms [ITU.G114.2003].

Lower Loss: For media transport it is important to minimise loss as it is usually not possible to retransmit within the delay budget for many connections. Whilst modern codecs can tolerate some loss it is beneficial to avoid it. The advantage of low delay congestion control is that since it aims to operate within the queuing boundaries it generally avoids loss.

Smoothness: The media rate should aim to be smooth within the constraints of the media, codec, and the network path. A smooth rate generally provides for more palatable media consumption.

Fairness: The system should aim to be reasonably fair with itself and TCP flows. Initially we aim for self fairness, and we will aim to tackle TCP fairness when we have sufficiently robust loss-mode operation.

[Burst Management]: [Due in later rev] We are working on mechanisms to manage the bursty nature of media allowing it maintain a smoother quality.

[Loss-based mode]: [Due in later rev] We are working on mechanisms to allow the system to compete with loss-based congestion control and maintain throughput, though without additional network support it is understood that the delay (and loss) would be largely beyond control.

5. Design Outline

The DFlow scheme aims to primarily utilise delay measurements to drive the congestion control. It currently utilises some of the core aspects of TFRC, such as its rate based operation, utilisation of the TCP equation, and its rate smoothing. It also employs similar signalling mechanisms. However as the design evolves we expect that DFlow may diverge further from TFRC.

5.1. Delay Composition

The total end-to-end one-way delay (OWD) a packet incurs may be considered to consist of four elements; transmission (or serialisation), propagation, processing, and queuing delays. For our purposes the first three elements may be considered together as a largely static component, termed the base delay. The base delay generally does not change significantly unless the node is mobile or the underlying link alters due to something like a route change. The main dynamic element of the delay, which DFlow aims to utilise, is the queuing delay. Taken together with the base delay, the queuing delay provides an indication of the actual path latency and also provides an insight as to the level of congestion on the path.

5.2. Delay Measurement

The notional one-way delay is measured for each packet by comparing the sender and receiver timestamps. Whilst the clocks on the sender and receiver are unlikely to be synchronised, it is assumed that their offset is relatively constant as the clock skew is generally quite small. Thus the notional OWD may only be used in a relative context. The notional OWD is measured for each packet over two sampling periods; Firstly over the longer base_period (typically $10 \times \text{RTT}$) from which the minima are stored as the base_delay. And secondly it is sampled over a shorter period current_period (typically 50ms), which is also filtered, usually also using a minima filter, and stored as current_delay. The minima of the OWDs are used to reduce noise of the measurements, which can be beneficial in the case of variable link types such as wireless.

5.3. Congestion Detection

The delay-based detection algorithm, outlined in Figure 1, operates by comparing the `current_delay` to the `base_delay`, which gives an indication of the queuing delay. If it exceeds a set congestion detection threshold, `cd_thresh`, then the packet is considered for the next stage of detection. The `cd_thresh` sets the limit for the queuing delay incurred by the flow, and is typically set at 50ms (we are also investigating automated thresholds). Once a flow has exceeded its `cd_thresh` then it undergoes a second test which is based upon the gradient of the delay change over two `current_period`'s, indicating that delay is on the increase, if it is positive then a 'congestion event' is flagged.

```
If ((base_delay - current_delay) > cd_thresh AND
    (current_delay - prev_current_delay) > 0)
    DelayCongestionEvent = True
```

Figure 1: Congestion Detection pseudo-code

This algorithm then provides input to the 'congestion interval history' mechanism (based on TFRC's 'loss interval history'), which is combined with normal input from the TFRC packet loss detection mechanisms, from which a 'congestion event rate' is derived which is then fed into the TCP equation to determine the send rate.

Note that we currently disable TFRC's oscillation reduction mechanism from [RFC5348] (Section 4.5) as it adversely affects the delay-based operation.

We have performed a number of simulations of the above mechanism in operation and have found it to be reasonably fair to itself, providing for smooth rates at suitable RTTs.

5.4. Slow Start

The delay based congestion detection is not only used during normal the congestion avoidance phase of the protocol but it also employed during slow start allowing for rapid, lower loss, attainment of the operating rate.

5.5. Loss-mode

We are actively investigating techniques to enable competitive behaviours with loss-based protocol such as TCP. We aim to develop a solution that provides for automatic fallback between loss and delay modes.

6. Further Work

The design is still under active development and there is more work to be done. We are seeking feedback on these ideas and future directions.

7. IANA Considerations

This document makes no requests of IANA.

8. Security Considerations

With a congestion control algorithm an attacker can attempt to interfere with the protocol to cause rate changes. However encryption of the protocol will largely protect it against such threats.

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.
- [RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, April 2007.
- [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

- [BudziszCxTCP.2011] Budzisz, L., Stanojevic, R., Schlote, A., Shorten, R., and F. Baker, "On the Fair Coexistence of Loss- and Delay-Based TCP", July 2011.
- [ChoiTFWC.2007] Choi, S. and M. Handley, "Fairer TCP-friendly congestion control protocol for multimedia streaming applications", Dec 2007.
- [GhanbariFuzzy.2007] Jammeh, E., Fleury, M., and M. Ghanbari, "Delay-based congestion avoidance for video communication with fuzzy logic control", Nov 2007.

- [I-D.nichols-tsvwg-codel]
Nichols, K. and V. Jacobson, "Controlled Delay Active Queue Management", draft-nichols-tsvwg-codel-01 (work in progress), February 2013.
- [ITU.G114.2003]
International Telecommunications Union, "One-way transmission time", ITU-T Recommendation G.707, May 2003.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, December 2012.
- [SisalemLDA.2000]
Sisalem, D. and A. Wolisz, "LDA+: A TCP-Friendly Adaptation Scheme for Multimedia Communication", May 2000.

Authors' Addresses

Piers O'Hanlon
University of Oxford
Oxford Internet Institute
1 St Giles
Oxford OX1 3JS
United Kingdom

Email: piers.ohanlon@oii.ox.ac.uk

Ken Carlberg
G11
1600 Clarendon Blvd
Arlington VA
USA

Email: carlberg@g11.org.uk

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 03, 2015

C. S. Perkins
University of Glasgow
July 02, 2014

Using RTP Control Protocol (RTCP) Feedback for Unicast Multimedia
Congestion Control
draft-perkins-rmcat-rtp-cc-feedback-01

Abstract

This memo discusses the types of congestion control feedback that it is possible to send using the RTP Control Protocol (RTCP), and their suitability of use in implementing congestion control for unicast multimedia applications.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 03, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Possible Models for RTCP Feedback	2
3. What Feedback is Achievable With RTCP?	4
3.1. Per-packet Feedback	4
3.2. Per-frame Feedback	4
3.3. Per-RTT Feedback	6
4. Discussion and Conclusions	6
5. Security Considerations	7
6. IANA Considerations	7
7. Acknowledgements	7
8. Informative References	7
Author's Address	8

1. Introduction

The coming deployment of WebRTC systems raises the prospect that high quality video conferencing will see extremely wide use. To ensure the stability of the network in the face of this use, WebRTC systems will need to use some form of congestion control for their RTP-based media traffic. To develop such congestion control, it is necessary to understand the sort of congestion feedback that can be provided within the framework of RTP [RFC3550] and the RTP Control Protocol (RTCP). It then becomes possible to determine if this is sufficient for congestion control, or if some form of RTP extension is needed.

This memo considers the congestion feedback that can be sent using RTCP under the RTP/SAVPF profile [RFC5124] (the secure version of the RTP/AVPF profile [RFC4585]). This profile was chosen as it forms the basis for media transport in WebRTC [I-D.ietf-rtcweb-rtp-usage] systems. Nothing in this memo is specific to the secure version of the profile, or to WebRTC, however.

2. Possible Models for RTCP Feedback

Several questions need to be answered when providing RTCP reception quality feedback for congestion control purposes. These include:

- o How often is feedback needed?
- o How much overhead is acceptable?
- o How much, and what, data does each report contain?

The key question is how often does the receiver need to send feedback on the reception quality it is experiencing, and hence the congestion

state of the network? Traditional congestion control protocols, such as TCP, send acknowledgements with every packet (or, at least, every couple of packets). That is straight-forward and low overhead when traffic is bidirectional and acknowledgements can be piggybacked onto return path data packets. It can also be acceptable, and can have reasonable overhead, to send separate acknowledgement packets when those packets are much smaller than data packets. It becomes a problem, however, when there is no return traffic on which to piggyback acknowledgements, and when acknowledgements are similar in size to data packets; this can be the case for some forms of media traffic, especially for voice over IP (VoIP) flows, but less so for video.

When considering multimedia traffic, it might make sense to consider less frequent feedback. For example, it might be possible to send a feedback packet once per video frame, or once per network round trip time (RTT). This could still give sufficiently frequent feedback for the congestion control loop to be stable and responsive while keeping the overhead reasonable when the feedback cannot be piggybacked onto returning data. In this case, it is important to note that RTCP can send much more detailed feedback than simple acknowledgements. For example, if it were useful, it could be possible to use an RTCP extended report (XR) packet [RFC3611] to send feedback once per RTT comprising a bitmap of lost and received packets, with reception times, over that RTT. As long as feedback is sent frequently enough that the control loop is stable, and the sender is kept informed when data leaves the network (to provide an equivalent to ACK clocking in TCP), it is not necessary to report on every packet at the instant it is received (indeed, it is unlikely that a video codec can react instantly to a rate change anyway, and there is little point in providing feedback more often than the codec can adapt).

The amount of overhead due to congestion control feedback that is considered acceptable has to be determined. RTCP data is sent in separate packets to RTP data, and this has some cost in terms of additional header overhead compared to protocols that piggyback feedback on return path data packets. The RTP standards have long said that a 5% overhead for RTCP traffic generally acceptable, while providing the ability to change this fraction. Is this still the case for congestion control feedback? Or is there a desire to either see more responsive feedback and congestion control, possibly with a higher overhead, or is lower overhead wanted, accepting that this might reduce responsiveness of the congestion control algorithm?

Finally, the details of how much, and what, data is to be sent in each report will affect the frequency and/or overhead of feedback. There is a fundamental trade-off that the more frequently feedback packets are sent, the less data can be included in each packet to

keep the overhead constant. Does the congestion control need high rate but simple feedback (e.g., like TCP acknowledgements), or is it acceptable to send more complex feedback less often?

3. What Feedback is Achievable With RTCP?

3.1. Per-packet Feedback

RTCP packets are sent as separate packets to RTP media data, and the protocol includes no mechanism for piggybacking an RTCP packet onto an RTP data packet. In addition, the RTCP timing rules are based on the size of the RTP session, the number of active senders, the RTCP packet size, and the configured RTCP bandwidth fraction, with randomisation to prevent synchronisation of reports; accordingly the RTCP packet transmission times are extremely unlikely to line up with RTP packet transmission times. As a result, RTCP cannot be used to send per-packet feedback in its current form.

All of these issues with using RTCP for per-packet feedback could be resolved in an update to the RTP protocol, of course. Such an update could change the RTCP timing rules, and might define a shim layer to allow multiplexing of RTP and RTCP into a single packet, or to extend the RTP header to piggyback feedback data. This sort of change would be a large, and almost certainly backwards incompatible, extension to the RTP protocol, and is unlikely to be completed quickly, but could be done if there was a need.

3.2. Per-frame Feedback

Consider one of the simplest scenarios for WebRTC: a point to point video call between two end systems. There will be four RTP flows in this scenario, two audio and two video, with all four flows being active for essentially all the time (the audio flows will likely use voice activity detection and comfort noise to reduce the packet rate during silent periods, and does not cause the transmissions to stop).

Assume all four flows are sent in a single RTP session, each using a separate SSRC. Further, assume each SSRC sends RTCP reports for all other SSRCs in the session (i.e., the optimisations in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] are not used, giving the worst case for the RTCP overhead). When all members are senders like this, the RTCP timing rules in Sections 6.2 and 6.3 of [RFC3550] and [RFC4585] reduce to:

$$\text{rtcp_interval} = \text{avg_rtcp_size} * n / \text{rtcp_bw}$$

where n is the number of members in the session, the `avg_rtcp_size` is measured in octets, and the `rtcp_bw` is the bandwidth available for RTCP, measured in octets per second (this will typically be 5% of the session bandwidth).

The average RTCP size will depend on the amount of feedback that is sent in each RTCP packet, on the number of members in the session, and on the size of source description (RTCP SDES) information sent. As a baseline, each RTCP packet will be a compound RTCP packet that contains an RTCP SR and an RTCP SDES packet. In the scenario above, each RTCP SR packet will contain three report blocks, once for each of the other RTP SSRCs sending data, for a total of 100 octets (this is 8 octets header, 20 octets sender info, and $3 * 24$ octets report blocks). The RTCP SDES packet will comprise a header (4 octets), an originating SSRC (4 octets), a CNAME chunk, and padding. If the CNAME follows [RFC7022] and [I-D.ietf-rtcweb-rtp-usage] it will be 19 octets in size, and require 1 octet of padding. The resulting compound RTCP packet will be 128 octets in size. If sent in UDP/IPv4 with no IP options and using Secure RTP, which adds 20 (IPv4) + 8 (UDP) + 14 (SRTP with 80 bit Authentication tag), the `avg_rtcp_size` will therefore be 170 octets, including the header overhead. The value n in this scenario is 4, and the `rtcp_bw` is assumed to be 5% of the session bandwidth.

If it is desired to send RTCP feedback packets on average 30 times per second, to correspond to one RTCP report every frame for 30fps video, one can invert the above `rtcp_interval` calculation to get an `rtcp_bw` that gives an interval of 1/30th of a second or lower. This corresponds to an `rtcp_bw` of 20400 octets per second (since $1/30 = 170 * 4 / 20400$). This is 163200 bits per second, which if 5% of the session bandwidth, gives a session bandwidth of approximately 3.3Mbps (i.e., 3.3Mbps media rate, plus an additional 5% for RTCP, to give a total data rate of approximately 3.4Mbps). That is, RTCP can report on every frame of video provided the session bandwidth is 3.3Mbps or larger, when every SSRC sends a report for every video frame (due to randomisation inherent in the RTCP timing rules, the actual RTCP transmission intervals will be within the range [0.0135, 0.0406]s, but will maintain an average RTCP transmission interval of 0.033s). This is not out of line with the expected session bandwidth for this type of application, suggesting the RTCP feedback can be used to provide per-frame congestion control feedback for WebRTC-style applications.

Note: To achieve the RTCP transmission intervals above the RTP/SAVPF profile with `T_rr_interval=0` is used, since even when using the reduced minimal transmission interval, the RTP/SAVPF profile would only allow sending RTCP at most every 0.11s (every third frame of video). Using RTP/SAVPF with `T_rr_interval=0` however is

capable of fully utilizing the configured 5% RTCP bandwidth fraction.

If additional feedback beyond the standard report block is needed, the session bandwidth needed will increase slightly. For example, with an additional 20 octets data being reported in each RTCP packet, the session bandwidth needed increases to 3.5Mbps for every SSRC to be able to report on every frame.

The above calculations highlight the baseline feasibility of RTCP congestion control feedback, but might not be the most appropriate usage of the RTCP bandwidth of all applications. Depending on needs, a less frequent usage of regular RTCP compound packets, controlled by `T_rr_interval` combined with using the reduced size RTCP packets, can achieve more frequent and useful reporting. Also the optimisations defined in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] will reduce the amount of bandwidth consumed for reporting when each endpoint has multiple SSRCs.

It might also seem unnecessary to assign the same fraction of the RTCP bandwidth to reporting on the audio and video, since video is much higher rate, and so is presumably more likely to cause congestion. Sending audio and video in separate RTCP sessions with their own RTCP bandwidth fraction would give essentially double the RTCP bandwidth for each video source, since RTCP bandwidth fraction would be shared between two reporting SSRCs, rather than between the four reporting SSRCs in the single session case. This would hence reduce the session bandwidth needed to allow reports on every frame. Extensions to split RTCP bandwidth unequally between participants in a single session could be defined to allow this to work with a single RTP session on a single UDP port, or two standard RTP sessions could be run on a single port, using a demultiplexing shim. RTCP already allows for different bandwidth fractions between senders and receivers, so this is a relatively small change to the protocol.

3.3. Per-RTT Feedback

The arguments made in Section 3.2 apply to this case as well. The network RTT will usually be larger than the media framing interval, so sending feedback per RTT is less of a load on RTCP than sending feedback per frame.

4. Discussion and Conclusions

RTCP as it is currently specified cannot be used to send per-packet congestion feedback. RTCP can, however, be used to send congestion feedback on each frame of video sent, provided the session bandwidth exceeds a couple of megabits per second (the exact rate depending on

the number of session participants, the RTCP bandwidth fraction, and whether audio and video are sent in one or two RTP sessions). RTCP can likely also be used to send feedback on a per-RTT basis, provided the RTT is not too low.

If it is desired to use RTCP in something close to its current form for congestion feedback in WebRTC, the multimedia congestion control algorithm needs to be designed to work with feedback sent roughly each frame or each RTT, rather than per packet, since that fits within the limitations of RTCP. That feedback can be a little more complex than just an acknowledgement, provided care is taken to consider the impact of the extra feedback on the overhead, possibly allowing for a degree of semantic feedback, meaningful to the codec layer as well as the congestion control algorithm.

Further study of the scenarios of interest is needed, to ensure that the analysis presented is applicable to other media topologies, and to sessions with different data rates and sizes of membership.

5. Security Considerations

The security considerations of [RFC3550], [RFC4585], and [RFC5124] apply.

6. IANA Considerations

There are no actions for IANA.

7. Acknowledgements

Thanks to Magnus Westerlund for his feedback on Section 3.2.

8. Informative References

[I-D.ietf-avtcore-rtp-multi-stream-optimisation]
Lennox, J., Westerlund, M., Wu, W., and C. Perkins,
"Sending Multiple Media Streams in a Single RTP Session:
Grouping RTCP Reception Statistics and Other Feedback",
draft-ietf-avtcore-rtp-multi-stream-optimisation-02 (work
in progress), February 2014.

[I-D.ietf-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-15 (work in progress), May
2014.

- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 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.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, September 2013.

Author's Address

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

Email: csp@csp Perkins.org

RMCAT WG
Internet-Draft
Intended status: Informational
Expires: April 23, 2014

V. Singh
J. Ott
Aalto University
October 20, 2013

Evaluating Congestion Control for Interactive Real-time Media
draft-singh-rmcat-cc-eval-04

Abstract

The Real-time Transport Protocol (RTP) is used to transmit media in telephony and video conferencing applications. This document describes the guidelines to evaluate new congestion control algorithms for interactive point-to-point real-time media.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Metrics	3
3.1. RTP Log Format	5
4. Guidelines	5
4.1. Avoiding Congestion Collapse	5
4.2. Stability	5
4.3. Media Traffic	6
4.4. Start-up Behaviour	6
4.5. Diverse Environments	6
4.6. Varying Path Characteristics	7
4.7. Reacting to Transient Events or Interruptions	7
4.8. Fairness With Similar Cross-Traffic	7
4.9. Impact on Cross-Traffic	7
4.10. Extensions to RTP/RTCP	8
5. Minimum Requirements for Evaluation	8
6. Evaluation Parameters	8
6.1. Bottleneck Traffic Flows	8
6.2. Access Links	9
6.3. Example Bottleneck Link Parameters	9
6.4. DropTail Router Queue Parameters	10
6.5. Media Flow Parameters	11
6.6. Cross-traffic Parameters	11
7. Status of Proposals	11
8. Security Considerations	12
9. IANA Considerations	12
10. Contributors	12
11. Acknowledgements	12
12. References	12
12.1. Normative References	12
12.2. Informative References	13
Appendix A. Application Trade-off	14
A.1. Measuring Quality	14
Appendix B. Proposal to evaluate Self-fairness of RMCAT congestion control algorithm	14
B.1. Evaluation Parameters	15
B.1.1. Media Traffic Generator	15
B.1.2. Bottleneck Link Bandwidth	16
B.1.3. Bottleneck Link Queue Type and Length	16
B.1.4. RMCAT flows and delay legs	16
B.1.5. Impairment Generator	17
B.2. Proposed Passing Criteria	17
B.3. Extensibility of the Experiment	17
Appendix C. Change Log	18
C.1. Changes in draft-singh-rmcat-cc-eval-04	18
C.2. Changes in draft-singh-rmcat-cc-eval-03	18

C.3. Changes in draft-singh-rmcat-cc-eval-02	18
C.4. Changes in draft-singh-rmcat-cc-eval-01	18
Authors' Addresses	19

1. Introduction

This memo describes the guidelines to help with evaluating new congestion control algorithms for interactive point-to-point real time media. The requirements for the congestion control algorithm are outlined in [I-D.jesup-rmcat-reqs]). This document builds upon previous work at the IETF: Specifying New Congestion Control Algorithms [RFC5033] and Metrics for the Evaluation of Congestion Control Algorithms [RFC5166].

The guidelines proposed in the document are intended to help prevent a congestion collapse, promote fair capacity usage and optimize the media flow's throughput. Furthermore, the proposed algorithms are expected to operate within the envelope of the circuit breakers defined in [I-D.ietf-avtcore-rtp-circuit-breakers].

This document only provides broad-level criteria for evaluating a new congestion control algorithm and the working group should expect a thorough scientific study to make its decision. The results of the evaluation are not expected to be included within the internet-draft but should be cited in the document.

2. Terminology

The terminology defined in RTP [RFC3550], RTP Profile for Audio and Video Conferences with Minimal Control [RFC3551], RTCP Extended Report (XR) [RFC3611], Extended RTP Profile for RTCP-based Feedback (RTP/AVPF) [RFC4585] and Support for Reduced-Size RTCP [RFC5506] apply.

3. Metrics

[RFC5166] describes the basic metrics for congestion control. Metrics that are of interest for interactive multimedia are:

- o Throughput.
- o Minimizing oscillations in the transmission rate (stability) when the end-to-end capacity varies slowly.
- o Delay.
- o Reactivity to transient events.

- o Packet losses and discards.
- o Section 2.1 of [RFC5166] discusses the tradeoff between throughput, delay and loss.

Each experiment is expected to log every incoming and outgoing packet (the RTP logging format is described in Section 3.1). The logging can be done inside the application or at the endpoints using pcap (packet capture, e.g., tcpdump, wireshark). The following are calculated based on the information in the packet logs:

1. Sending rate, Receiver rate, Goodput
2. Packet delay
3. Packet loss
4. If using, retransmission or FEC: residual loss
5. Packets discarded from the playout or de-jitter buffer

[Open issue (1): The "unfairness" test is (measured at 1s intervals):

1. Do not trigger the circuit breaker.
2. Over 3 times or less than 1/3 times the throughput for an RMCAT media stream compared to identical RMCAT streams competing on a bottleneck, for a case when the competing streams have similar RTTs.
3. Over 3 times delay compared to RTT measurements performed before starting the RMCAT flow or for the case when competing with identical RMCAT streams having similar RTTs.

]

[Open issue (2): Possibly using Jain-fairness index.]

Convergence time: the time taken to reach a stable rate at startup, after the available link capacity changes, or when new flows get added to the bottleneck link.

Bandwidth Utilization, defined as ratio of the instantaneous sending rate to the instantaneous bottleneck capacity. This metric is useful when an RMCAT flow is by itself or competing with similar cross-traffic.

From the logs the statistical measures (min, max, mean, standard deviation and variance) for the whole duration or any specific part of the session can be calculated. Also the metrics (sending rate, receiver rate, goodput, latency) can be visualized in graphs as variation over time, the measurements in the plot are at 1 second intervals. Additionally, from the logs it is possible to plot the histogram or CDF of packet delay.

3.1. RTP Log Format

The log file is tab or comma separated containing the following details:

```
Send or receive timestamp (unix)
RTP payload type
SSRC
RTP sequence no
RTP timestamp
marker bit
payload size
```

If the congestion control implements, retransmissions or FEC, the evaluation should report both packet loss (before applying error-resilience) and residual packet loss (after applying error-resilience).

4. Guidelines

A congestion control algorithm should be tested in simulation or a testbed environment, and the experiments should be repeated multiple times to infer statistical significance. The following guidelines are considered for evaluation:

4.1. Avoiding Congestion Collapse

The congestion control algorithm is expected to take an action, such as reducing the sending rate, when it detects congestion. Typically, it should intervene before the circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] is engaged.

Does the congestion control propose any changes to (or diverge from) the circuit breaker conditions defined in [I-D.ietf-avtcore-rtp-circuit-breakers].

4.2. Stability

The congestion control should be assessed for its stability when the path characteristics do not change over time. Changing the media encoding rate estimate too often or by too much may adversely affect the application layer performance.

4.3. Media Traffic

The congestion control algorithm should be assessed with different types of media behavior, i.e., the media should contain idle and data-limited periods. For example, periods of silence for audio, varying amount of motion for video, or bursty nature of I-frames.

The evaluation may be done in two stages. In the first stage, the endpoint generates traffic at the rate calculated by the congestion controller. In the second stage, real codecs or models of video codecs are used to mimic application-limited data periods and varying video frame sizes.

4.4. Start-up Behaviour

The congestion control algorithm should be assessed with different start-rates. The main reason is to observe the behavior of the congestion control in different evaluation scenarios, such as when competing with varying amount of cross-traffic or how quickly does the congestion control algorithm achieve a stable sending rate.

[Editor's note: requires a robust definition for unfriendliness and convergence time.]

4.5. Diverse Environments

The congestion control algorithm should be assessed in heterogeneous environments, containing both wired and wireless paths. Examples of wireless access technologies are: 802.11, GPRS, HSPA, or LTE. One of the main challenges of the wireless environments for the congestion control algorithm is to distinguish between congestion induced loss and transmission (bit-error corruption) loss. Congestion control algorithms may incorrectly identify transmission loss as congestion loss and reduce the media encoding rate by too much, which may cause oscillatory behavior and deteriorate the users' quality of experience. Furthermore, packet loss may induce additional delay in networks with wireless paths due to link-layer retransmissions.

4.6. Varying Path Characteristics

The congestion control algorithm should be evaluated for a range of path characteristics such as, different end-to-end capacity and latency, varying amount of cross traffic on a bottleneck link and a router's queue length. For the moment, only DropTail queues are used. However, if new Active Queue Management (AQM) schemes become available, the performance of the congestion control algorithm should be again evaluated.

In an experiment, if the media only flows in a single direction, the feedback path should also be tested with varying amounts of impairments.

The main motivation for the previous and current criteria is to identify situations in which the proposed congestion control is less performant.

4.7. Reacting to Transient Events or Interruptions

The congestion control algorithm should be able to handle changes in end-to-end capacity and latency. Latency may change due to route updates, link failures, handovers etc. In mobile environment the end-to-end capacity may vary due to the interference, fading, handovers, etc. In wired networks the end-to-end capacity may vary due to changes in resource reservation.

4.8. Fairness With Similar Cross-Traffic

The congestion control algorithm should be evaluated when competing with other RTP flows using the same or another candidate congestion control algorithm. The proposal should highlight the bottleneck capacity share of each RTP flow.

[Editor's note: If we define Unfriendliness then that criteria should be applied here.]

4.9. Impact on Cross-Traffic

The congestion control algorithm should be evaluated when competing with standard TCP. Short TCP flows may be considered as transient events and the RTP flow may give way to the short TCP flow to complete quickly. However, long-lived TCP flows may starve out the RTP flow depending on router queue length.

The proposal should also measure the impact on varied number of cross-traffic sources, i.e., few and many competing flows, or mixing various amounts of TCP and similar cross-traffic.

4.10. Extensions to RTP/RTCP

The congestion control algorithm should indicate if any protocol extensions are required to implement it and should carefully describe the impact of the extension.

5. Minimum Requirements for Evaluation

[Editor's Note: If needed, a minimum evaluation criteria can be based on the above guidelines or defined tests/scenarios.]

6. Evaluation Parameters

An evaluation scenario is created from a list of network, link and flow characteristics. The example parameters discussed in the following subsections are meant to aid in creating evaluation scenarios and do not describe an evaluation scenario. The scenario discussed in Appendix B takes into account all these parameters.

6.1. Bottleneck Traffic Flows

The network scenario describes the types of flows sharing the common bottleneck with a single RMCAT flow, they are:

1. A single RMCAT flow by itself.
2. Competing with similar RMCAT flows. These competing flows may use the same algorithm or another candidate RMCAT algorithm.
3. Compete with long-lived TCP.
4. Compete with bursty TCP.
5. Compete with LEDBAT flows.
6. Compete with unresponsive interactive media flows (i.e., not only CBR).

Figure 1 shows an example evaluation topology, where S1..Sn are traffic sources, these sources are either RMCAT or a mixture of traffic flows listed above. R1..Rn are the corresponding receivers. A and B are routers that can be configured to introduce impairments. Access links are in between the sender/receiver and the router, while the bottleneck link is between the Routers A and B.

```
+---+ Access                               Access +---+
|S1 |===== \                             / =====|R1 |
+---+ link \\                             // link +---+
```

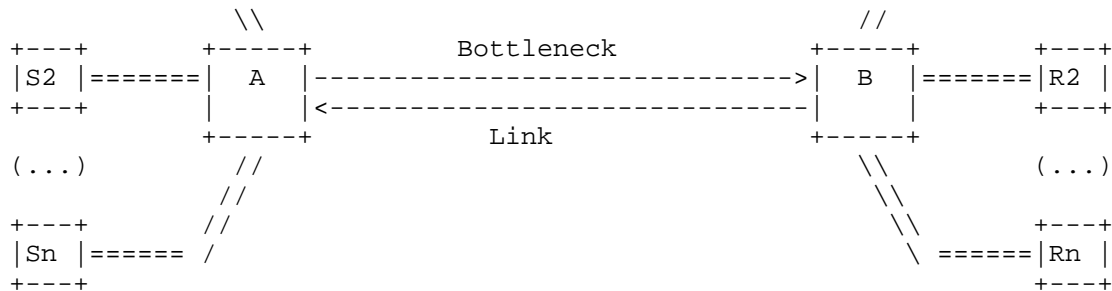


Figure 1: Simple Topology

[Open Issue: Discuss more complex topologies]

6.2. Access Links

The media senders and receivers are typically connected to the bottleneck link, common access links are:

1. Ethernet (LAN)
2. Wireless LAN (WLAN)
3. 3G/LTE

[Open issue: point to a reference containing parameters or traces to model WLAN and 3G/LTE.]

A real-world network typically consists of a mixture of links, the most important aspect is to identify the location of the bottleneck link. The bottleneck link can move from one node to another depending on the amount of cross-traffic or due to the varying link capacity. The design of the experiments should take this into account. In the simplest case the access link may not be the bottleneck link but an intermediate node.

6.3. Example Bottleneck Link Parameters

The bottleneck link carries multiple flows, these flows may be other RMCAT flows or other types of cross-traffic. The experiments should dimension the bottleneck link based on the number of flows and the expected behavior. For example, if 5 media flows are expected to share the bottleneck link equally, the bottleneck link is set to 5 times the desired transmission rate.

If the experiment carries only media in one direction, then the upstream (sender to receiver) bottleneck link carries media packets

while the downstream (receiver to sender) bottleneck carries the feedback packets. The bottleneck link parameters discussed in this section apply only to a single direction, hence the bottleneck link in the reverse direction can choose the same or have different parameters.

The link latency corresponds to the propagation delay of the link, i.e., the time it takes for a packet to traverse the bottleneck link, it does not include queuing delay. In an experiment with several links the experiment should describe if the links add latency or not. It is possible for experiments to have multiple hops with different link latencies. Experiments are expected to verify that the congestion control is able to work in challenging situations, for example over trans-continental and/or satellite links. The experiment should pick link latency values from the following:

1. Very low latency: 0-1ms
2. Low latency: 50ms
3. High latency: 150ms
4. Extreme latency: 300ms

Similarly, to model lossy links, the experiments can choose one of the following loss rates, the fractional loss is the ratio of packets lost and packets sent.

1. no loss: 0%
2. 1%
3. 5%
4. 10%
5. 20%

These fractional losses can be generated using traces, Gilbert-Elliot model, randomly (uncorrelated) loss.

6.4. DropTail Router Queue Parameters

The router queue length is measured as the time taken to drain the FIFO queue, they are:

1. QoS-aware (or short): 70ms

2. Nominal: 500ms

3. Buffer-bloated: 2000ms

However, the size of the queue is typically measured in bytes or packets and to convert the queue length measured in seconds to queue length in bytes:

$$\text{QueueSize (in bytes)} = \text{QueueSize (in sec)} \times \text{Throughput (in bps)} / 8$$

6.5. Media Flow Parameters

The media sources can be modeled in two ways. In the first, the sources always have data to send, i.e., have no data limited intervals and are able to generate the media rate requested by the RMCAT congestion control algorithm. In the second, the traffic generator models the behavior of a media codec, mainly the burstiness (time-varying data produced by a video GOP).

At the beginning of the session, the media sources are configured to start at a given start rate, they are:

1. 200 kbps

2. 800 kbps

3. 1300 kbps

4. 4000 kbps

6.6. Cross-traffic Parameters

Long-lived TCP flows will download data throughout the session and are expected to have infinite amount of data to send or receive.]

[Open issue: short-lived/bursty TCP cross-traffic parameters are still TBD.

7. Status of Proposals

Congestion control algorithms are expected to be published as "Experimental" documents until they are shown to be safe to deploy. An algorithm published as a draft should be experimented in simulation, or a controlled environment (testbed) to show its applicability. Every congestion control algorithm should include a note describing the environments in which the algorithm is tested and safe to deploy. It is possible that an algorithm is not recommended for certain environments or perform sub-optimally for the user.

[Editor's Note: Should there be a distinction between "Informational" and "Experimental" drafts for congestion control algorithms in RMCAT. [RFC5033] describes Informational proposals as algorithms that are not safe for deployment but are proposals to experiment with in simulation/testbeds. While Experimental algorithms are ones that are deemed safe in some environments but require a more thorough evaluation (from the community).]

8. Security Considerations

Security issues have not been discussed in this memo.

9. IANA Considerations

There are no IANA impacts in this memo.

10. Contributors

The content and concepts within this document are a product of the discussion carried out in the Design Team.

Michael Ramalho provided the text for the scenario discussed in Appendix B.

11. Acknowledgements

Much of this document is derived from previous work on congestion control at the IETF.

The authors would like to thank Harald Alvestrand, Luca De Cicco, Wesley Eddy, Lars Eggert, Kevin Gross, Vinayak Hegde, Stefan Holmer, Randell Jesup, Piers O'Hanlon, Colin Perkins, Michael Ramalho, Zaheduzzaman Sarker, Timothy B. Terriberry, Michael Welzl, and Mo Zanaty for providing valuable feedback on earlier versions of this draft. Additionally, also thank the participants of the design team for their comments and discussion related to the evaluation criteria.

12. References

12.1. Normative References

- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

- [RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 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.
- [I-D.jesup-rmcat-reqs]
Jesup, R., "Congestion Control Requirements For RMCAT", draft-jesup-rmcat-reqs-01 (work in progress), February 2013.
- [I-D.ietf-avtcore-rtp-circuit-breakers]
Perkins, C. and V. Singh, "RTP Congestion Control: Circuit Breakers for Unicast Sessions", draft-ietf-avtcore-rtp-circuit-breakers-01 (work in progress), October 2012.

12.2. Informative References

- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, August 2007.
- [RFC5166] Floyd, S., "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166, March 2008.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [SA4-EVAL]
R1-081955, 3GPP., "LTE Link Level Throughput Data for SA4 Evaluation Framework", 3GPP R1-081955, 5 2008.
- [SA4-LR]
S4-050560, 3GPP., "Error Patterns for MBMS Streaming over UTRAN and GERAN", 3GPP S4-050560, 5 2008.
- [TCP-eval-suite]
Lachlan, A., Marcondes, C., Floyd, S., Dunn, L., Guillier, R., Gang, W., Eggert, L., Ha, S., and I. Rhee, "Towards a Common TCP Evaluation Suite", Proc. PFLDnet. 2008, August 2008.

Appendix A. Application Trade-off

Application trade-off is yet to be defined. see RMCAT requirements [I-D.jesup-rmcat-reqs] document. Perhaps each experiment should define the application's expectation or trade-off.

A.1. Measuring Quality

No quality metric is defined for performance evaluation, it is currently an open issue. However, there is consensus that congestion control algorithm should be able to show that it is useful for interactive video by performing analysis using a real codec and video sequences.

Appendix B. Proposal to evaluate Self-fairness of RMCAT congestion control algorithm

The goal of the experiment discussed in this section is to initially take out as many unknowns from the scenario. Later experiments can define more complex environments, topologies and media behavior. This experiment evaluates the performance of the RMCAT sender competing with other similar RMCAT flows (running the same algorithm or other RMCAT proposals) on the bottleneck link. There are up to 20 RMCAT flows competing for capacity, but the media only flows in one direction, from senders (S1..S20) to receivers (R1..R20) and the feedback packets flow in the reverse direction.

Figure 2 shows the experiment setup and it has subtle differences compared to the simple topology in Figure 1. Groups of 10 receivers are connected to the bottleneck link through two different routers (Router C and D). The rationale for adding these additional routers is to create two delay legs, i.e., two groups of endpoints with different network latencies and measure the performance of the RMCAT congestion control algorithm. If fewer than 10 sources are initialized, all traffic flows experience the same delay because they share the same delay leg.

Router A has a single forward direction bottleneck link (i.e., the bottleneck capacity and delay constraints applies only to the media packets going from the sender to the receiver, the feedback packets are unaffected). Hence, the Round-Trip Time (RTT) is primarily composed of the bottleneck queue delay and any forward path (propagation) latency. The main reason for not applying any constraints on the return path is to provide the best-case performance scenario for the congestion control algorithm. In later experiments, it is possible to add similar capacity and delay constraints on the return path.

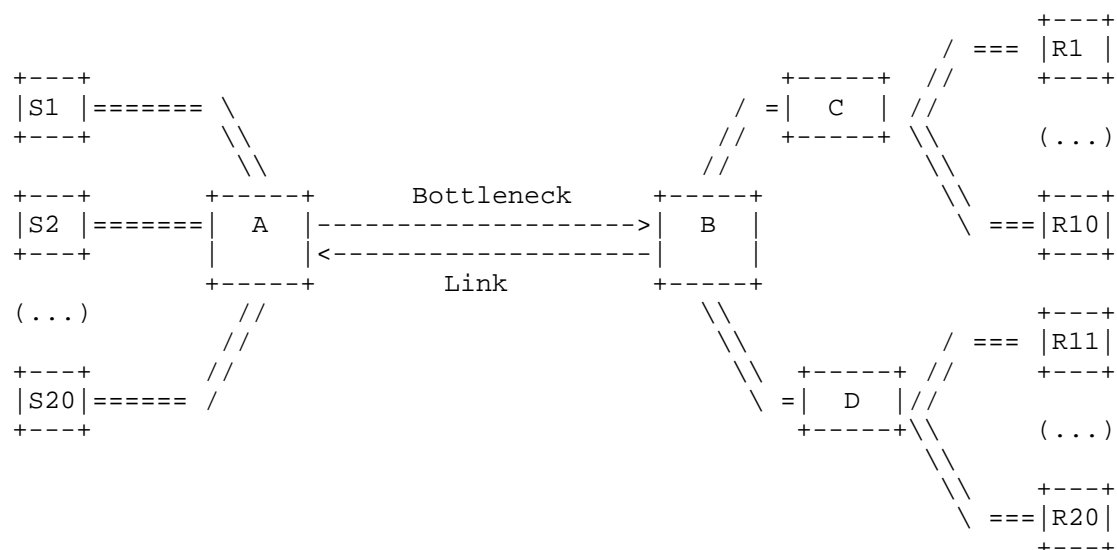


Figure 2: Self-fairness Evaluation Setup

Loss impairments are applied at Router C and Router D, but only to the feedback flows. If the losses are set to 0%, it represents a case where the return path is over-provisioned for all traffic. In later experiments the loss impairments can be added to the media path as well.

The media sources are configured to send infinite amount of data, i.e., the sources always have data to send and have no data limited intervals. Additionally, the media sources are always successful in generating the media rate requested by the RMCAT congestion control algorithm. In this experiment, we avoid the potentially complicated scenario of using media traffic generators that try to model the behavior of media codecs (mainly the burstiness).

B.1. Evaluation Parameters

B.1.1. Media Traffic Generator

The media source always generates at the rate requested by the congestion control and has infinite data to send. Furthermore, the media packet generator is subject to the following constraints:

1. It MUST emit a packet at least once per 100 ms time interval.
2. For low media rate source: when generating data at a rate less than a maximum length MTU every 100 ms would allow (e.g., 120

kbps = 1500 bytes/packet * 10 packets/sec * 8 bits/byte), the RMCAT source must modulate the packet size (RTP payload size) of RTP packets that are sent every 100 ms to attain the desired rate.

3. For high media rate sources: when generating data at a rate greater than a maximum length MTU every 100 ms would allow, the source must do so by sending (approximately) maximum MTU sized packets and adjusting the inter-departure interval to be approximately equal. The intent of this is to ensure the data is sent relatively smoothly independent of the bit rate, subject to the first constraint.

B.1.2. Bottleneck Link Bandwidth

The bottleneck link capacity is dimensioned such that each RMCAT flow in an ideal situation with perfectly equal capacity sharing for all the flows on the bottleneck obtains the following throughputs: 200 kbps, 800 kbps, 1.3 Mbps and 4 Mbps.

For example, experiments with five RMCAT flows with an 800 kbps/flow target rate should set the bottleneck link capacity to 4 Mbps.

B.1.3. Bottleneck Link Queue Type and Length

The bottleneck link queue (Router A) is a simple FIFO queue having a buffer length corresponding to 70 ms, 500 ms or 2000 ms (defined in Section 6.4) of delay at the bottleneck link rate (i.e., actual buffer lengths in bytes are dependent on bottleneck link bandwidth).

B.1.4. RMCAT flows and delay legs

Experiments run with 1, 3, 5, 10 and 20 RMCAT sources, they are outlined as follows:

1. Experiments with 1, 3, and 5 RMCAT flows, all RMCAT flows commence simultaneously. A single delay leg is used and the link latency is set to one of the following : 0 ms, 50 ms and 150 ms.
2. For 10 and 20 source experiments where all RMCAT flows begin simultaneously the sources are split evenly into two different bulk delay legs. One leg is set to 0 ms bulk delay leg and the other is set to 150 ms.
3. For 10 and 20 source experiments where the first set will use 0 ms of bulk delay and the second set will use 150 ms bulk delay.
 1. Random starts within interval [0 ms, 500 ms].

2. One "early-coming" flow (i.e., the 1st flow starting and achieving steady-state before the next N-1 simultaneously begin).
3. One "late-coming" flow (i.e., the Nth flow starting after steady-state has occurred for the existing N-1 flows).

These cases assess if there are any early or late-comer advantages or disadvantages for a particular algorithm and to see if any unfairness is reproducible or unpredictable.

[Open issue (A.1): which group does the early and late flow belong to?]

[Open issue (A.2): Start rate for the media flows]

B.1.5. Impairment Generator

Packet loss is created in the reverse path (affects only feedback packets). Cases of 0%, 1%, 5% and 10% are studied for the 1, 3, and 5 RMCAT flow experiments, losses are not applied to flows with 10 or 20 RMCAT flows.

B.2. Proposed Passing Criteria

[Editor's note: there has been little or no discussion on the below criteria, however, they are listed here for the sake of completeness.]

No unfairness is observed, i.e., at steady state each flow attains a throughput between $[B/(3*N), (3*B)/N]$, where B is the link bandwidth and N is the number of flows.

No flow experiences packet loss when queue length is set to 500 ms or greater.

All individual sources must be in their steady state within twenty LRTTs (where LRTT is defined as the RTT associated with the flow with the Largest RTT in the experiment).]

B.3. Extensibility of the Experiment

The above scenario describes only RMCAT sources competing for capacity on the bottleneck link, however, future experiments can use different types of cross-traffic (as described in Section 6.1).

Currently, the forward path (carrying media packets) is characterized to add delay and a fixed bottleneck link capacity, in the future packet losses and capacity changes can be applied to mimic a wireless

link layer (for e.g., WiFi, 3G, LTE). Additionally, only losses are applied to the reverse path (carrying feedback packets), later experiments can apply the same forward path (carrying media packets) impairments to the reverse path.

Appendix C. Change Log

Note to the RFC-Editor: please remove this section prior to publication as an RFC.

C.1. Changes in draft-singh-rmcat-cc-eval-04

- o Incorporate feedback from IETF 87, Berlin.
- o Clarified metrics: convergence time, bandwidth utilization.
- o Changed fairness criteria to fairness test.
- o Added measuring pre- and post-repair loss.
- o Added open issue of measuring video quality to appendix.
- o clarified use of DropTail and AQM.
- o Updated text in "Minimum Requirements for Evaluation"

C.2. Changes in draft-singh-rmcat-cc-eval-03

- o Incorporate the discussion within the design team.
- o Added a section on evaluation parameters, it describes the flow and network characteristics.
- o Added Appendix with self-fairness experiment.
- o Changed bottleneck parameters from a proposal to an example set.

C.3. Changes in draft-singh-rmcat-cc-eval-02

- o Added scenario descriptions.

C.4. Changes in draft-singh-rmcat-cc-eval-01

- o Removed QoE metrics.
- o Changed stability to steady-state.
- o Added measuring impact against few and many flows.

- o Added guideline for idle and data-limited periods.
- o Added reference to TCP evaluation suite in example evaluation scenarios.

Authors' Addresses

Varun Singh
Aalto University
School of Electrical Engineering
Otakaari 5 A
Espoo, FIN 02150
Finland

Email: varun@comnet.tkk.fi
URI: <http://www.netlab.tkk.fi/~varun/>

Joerg Ott
Aalto University
School of Electrical Engineering
Otakaari 5 A
Espoo, FIN 02150
Finland

Email: jo@comnet.tkk.fi

RTP Media Congestion Avoidance
Techniques (rmcat)
Internet-Draft
Intended status: Experimental
Expires: July 23, 2013

M. Welzl
University of Oslo
January 19, 2013

Coupled congestion control for RTP media
draft-welzl-rmcat-coupled-cc-00

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 23, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

When there is enough data to send, a congestion controller must increase its sending rate until the path's available capacity has been reached; depending on the controller, sometimes the rate is increased further, until packets are ECN-marked or dropped. In the public Internet, this is currently the only way to get any feedback from the network that can be used as an indication of congestion. 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, however, 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 which 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. It is recognized that 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 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. It is passive in that it does not actively initiate communication with flows and the 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 is often different from what the congestion controller originally determined. Using information about/from the currently active flows, SBD updates the FSE with the correct Flow Group Identifiers (FGIs).

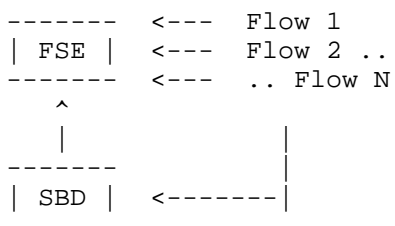


Figure 1: Coupled congestion control architecture

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 FGIs, 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 from measurements, by considering correlations among measured delay and loss as an indication of a shared bottleneck, or 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) are typically routed in the same way. The latter method is the only one specified in this document: SBD MUST consider all flows that use the same five-tuple 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]. Port numbers are needed as part of the classification due to mechanisms like Equal-Cost Multi-Path (ECMP) routing which use different paths for packets towards the same destination, but are typically configured to keep packets from the same transport connection on the same path.

5.2. FSE

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

- o a unique flow number to identify the flow
- o 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)
- o 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.
- o The calculated rate CR, i.e. the rate that was most recently calculated by the flow's congestion controller.
- 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 greedy flow, DR must be set to CR. A DR value that is larger than CR indicates that the flow has taken leftover bandwidth from a non-greedy flow.
- o S_CR, the sum of the calculated rates of all flows in the same FG (including the flow itself), as seen by the flow during its last rate update.

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 non-greedy flows, the same otherwise). UPDATE returns a rate that should be used instead of the rate that the congestion controller has determined.

Below, an example algorithm is described. While other algorithms could be used instead, the same algorithm must be applied to all flows. The way the algorithm is described here, the operations are carried out by the flows, but they are the same for all flows. This means that the algorithm could, for example, be implemented in a library that provides registration, deregistration functions and the UPDATE function. To minimize the number of changes to existing

applications, one could, however, also embed this functionality in the FSE element.

5.3.1. Example algorithm

- (1) When a flow starts, it registers itself with SBD and the FSE. CR and DR are initialized with the congestion controller's initial rate. SBD will assign the correct FGI. When a flow is assigned an FGI, its S_CR is initialized to be the sum of the calculated rates of all the flows in its FG.
- (2) When a flow stops, it sets its DR to 0 and negates P.
- (3) Every time the flow's congestion controller determines a new sending rate new_CR, assuming the flow's new desired rate new_DR to be "infinity" in case of a greedy flow with an unknown maximum rate, the flow calls UPDATE, which carries out the following tasks:
 - (a) For all the flows in its FG (including itself), it calculates the sum of all the absolute values of all priorities, S_P, the sum of all desired rates, S_DR, and the sum of all the calculated rates, new_S_CR.
 - (b) It updates CR if new_CR is smaller than the already stored CR value, or if new_S_CR is smaller or equal to the flow's stored S_CR value. This restriction on updating CR ensures that only one flow can make S_CR increase at a time.
 - (c) It updates new_S_CR using its own updated CR, and updates S_CR with new_S_CR.
 - (d) It subtracts DR from S_DR, updates DR to $\min(\text{new_DR}, \text{CR})$, and adds the updated DR to S_DR.
 - (e) It initializes the total leftover rate TLO to 0. Then, for every other flow i in its FG that has $\text{DR}(i) < \text{CR}(i)$, it calculates the leftover rate as $\text{abs}(P(i))/S_P * S_CR - \text{DR}(i)$, adds the flow's leftover rate to TLO, and sets $\text{DR}(i)$ to $\text{CR}(i)$. This makes flow i look like a greedy flow and ensures that the leftover rate can only once be taken from it. Finally, if $P(i)$ is negative, it removes flow i's entry from the FSE.
 - (f) It calculates the new sending rate as $\min(\text{new_DR}, P/S_P * S_CR + \text{TLO})$. This gives the flow the correct share of the bandwidth based on its priority, applies an upper bound in case of an application-limited flow, and adds any

potentially leftover bandwidth from non-greedy flows.

- (g) If the flow's new sending rate is greater than DR, then it updates DR with the flow's new sending rate.

The goals of the flow algorithm are to achieve prioritization, improve network utilization in the face of non-greedy 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 non-greedy 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.

There are more potential issues with the algorithm described here. Rule 3 b) leads to a conservative behavior: it ensures that only one flow at a time can increase the overall sending rate. This rule is probably appropriate for situations where minimizing delay is the major goal, but it may not fit for all purposes; it also does not incorporate the magnitude by which a flow can increase its rate. Notably, despite this limitation on the overall rate of all flows per FGI, immediate rate jumps of single flows could become problematic when the FSE is used in a highly asynchronous manner, e.g. when flows have very different RTTs. Rule 3 e) gives all the leftover rate of non-greedy 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.

5.3.2. Example operation

In order to illustrate the operation of the 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 greedy and considers itself to have top priority. This is the FSE after the flow algorithm's step 1:

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	1	1	1	1

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

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	10	10	10	10

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

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	10	10	10	10
2	1	0.5	1	1	11	1

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.

```

new_CR = 8. new_DR = infinity.
3 a) S_P = 1.5; S_DR = 11; new_S_CR = 11.
3 b) new_CR < CR, hence CR = 8.
3 c) new_S_CR = 9; S_CR = 9.
3 d) DR = CR = 8; S_DR = 9.
3 e) TLO = 0; there are no other flows with DR < CR.
3 f) new sending rate: min(infinity, 1/1.5 * 9 + 0) = 6.
3 g) does not apply.

```

The resulting FSE looks as follows:

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	8	8	9	6
2	1	0.5	1	1	11	1

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.

```

new_CR=2. new_DR = infinity.
3 a) S_P = 1.5; S_DR = 9; new_S_CR = 9.
3 b) new_CR > CR but new_S_CR < S_CR, hence CR = 2.
3 c) new_S_CR = 10; S_CR = 10.
3 d) DR = CR = 2; S_DR = 10.
3 e) TLO = 0; there are no other flows with DR < CR.
3 f) new sending rate: min(infinity, 0.5/1.5 * 10 + 0) = 3.33.
3 g) new sending rate > DR, hence DR = 3.33.

```

The resulting FSE looks as follows:

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	8	8	9	6
2	1	0.5	2	3.33	10	3.33

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.

new_CR=9. new_DR=2.

3 a) $S_P = 1.5$; $S_{DR} = 11.33$; new_S_CR = 10.

3 b) new_CR > CR and new_S_CR > S_CR, hence CR is not updated (since flow #2 has just increased S_CR, flow #1 cannot also increase it in this iteration).

3 c) new_S_CR = 10; S_CR = 10.

3 d) DR = 2; S_DR = 5.33.

3 e) TLO = 0; there are no other flows with DR < CR.

3 f) new sending rate: $\min(2, 1/1.5 * 10 + 0) = 2$. Note that, without the 2 Mbit/s limitation from the application, the new sending rate for flow #1 would now be 6.66 Mbit/s, leading to perfect network saturation ($6.66 + 3.33 = \text{approx. } 10$).

3 g) does not apply.

The resulting FSE looks as follows:

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	8	2	10	2
2	1	0.5	2	3.33	10	3.33

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.

```

new_CR=3. new_DR = infinity.
3 a) S_P = 1.5; S_DR = 5.33; new_S_CR = 10.
3 b) new_CR > CR but new_S_CR = S_CR, hence CR = 3.
3 c) new_S_CR = 11; S_CR = 11.
3 d) DR = 3; S_DR = 5.
3 e) TLO = 0; flow #1 has DR < CR, hence TLO += 1/1.5 * 11
    - 2 = 5.33.
DR of flow #1 is set to 8. Flow #1 does not have a negative
P(i) value, so its entry is not deleted.
3 f) new sending rate: min(infinity, 0.5/1.5*11 + 5.33) = 9.
3 g) new sending rate > DR, hence DR = 9.

```

The resulting FSE looks as follows:

#	FGI	P	CR	DR	S_CR	Rate
1	1	1	8	8	10	2
2	1	0.5	3	9	11	9

Now, the total rate of the two flows is $2 + 9 = 11$ Mbit/s, exceeding the total capacity by the 1 Mbit/s by which the congestion controller of flow #2 has increased its rate. Note that, had flow #1 been greedy, the same total rate would have resulted after this iteration. 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.

```

new_CR = 1. new_DR = infinity.
3 a) S_P = 1.5; S_DR = 9; new_S_CR = 11.
3 b) new_CR < CR hence CR = 1.
3 c) new_S_CR = 9; S_CR = 9.
3 d) DR = 1; S_DR = 1.
3 e) TLO = 0; flow #1 has DR < CR, hence TLO += 1/1.5 * 9 - 0 = 6.
DR of flow #1 is set to 8. Flow #1 has a negative P(i) value, so
its entry is deleted.
3 f) new sending rate: min(infinity, 0.5/1.5 * 9 + 6) = 9.
3 g) new sending rate > DR, hence DR = 9.

```

The resulting FSE looks as follows:

#	FGI	P	CR	DR	S_CR	Rate
1	1	-1	8	0	10	2
2	1	0.5	1	9	9	9

(before deletion)

Now, the total rate, used only by flow #2, is 9 Mbit/s, which is the rate that it would have had alone upon reacting to congestion after a sending rate of 11 Mbit/s.

6. Acknowledgements

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

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.

9.2. Informative References

- [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-05.txt (work in progress), October 2012.
- [rtcweb-usecases]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-10.txt (work in progress), December 2012.
- [transport-multiplex]
Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing-04.txt (work in progress), October 2012.

Author's Address

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

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

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

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. System Model	3
4. NADA Receiver Behavior	4
4.1 Estimation of one-way delay and queuing delay	4
4.2 Estimation of packet loss/marketing ratio	5
4.3 Non-linear warping of delay	6
4.4 Aggregating congestion signals	7
4.5 Estimating receiving rate	7
4.6 Sending periodic feedback	7
4.7 Discussions on delay metrics	8
5. NADA Sender Behavior	9
5.1 Reference rate calculation	10
5.1.1 Accelerated ramp up	10
5.1.2. Gradual rate update	11
5.2 Video encoder rate control	12
5.3 Rate shaping buffer	12
5.4 Adjusting video target rate and sending rate	12
6. Incremental Deployment	13
7. Implementation Status	13
8. IANA Considerations	14
9. References	14
9.1 Normative References	14
9.2 Informative References	14
Appendix A. Network Node Operations	15
A.1 Default behavior of drop tail	16
A.2 ECN marking	16
A.3 PCN marking	16
Authors' Addresses	17

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/marketing 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:

$$\begin{aligned} d_{\text{tilde}_n} &= (d_{\text{hat}_n}), \quad \text{if } d_{\text{hat}_n} < d_{\text{th}}; \\ d_{\text{tilde}_n} &= d_{\text{th}} \frac{(d_{\text{max}} - d_{\text{hat}_n})^4}{(d_{\text{max}} - d_{\text{th}})^4}, \quad \text{if } d_{\text{th}} < d_{\text{hat}_n} < d_{\text{max}}; \\ d_{\text{tilde}_n} &= 0, \quad \text{if } d_{\text{hat}_n} > d_{\text{max}}. \end{aligned}$$

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 $(0, d_{\text{th}})$, NADA operates in pure delay-based mode if no losses/markings are present. When queuing delay exceeds d_{max} , NADA reacts to loss/markings 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:

$$x_n = d_{\text{tilde}_n} + p_M d_M + p_L d_L, \quad (1)$$

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 $\langle d_{\text{hat}_n}, x_n, x'_n, R_r \rangle$ 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:

$$x'_n = \frac{x_n - x_{(n-k)}}{\text{delta}}, \quad (2)$$

where the interval between consecutive RTCP feedback messages is denoted as δ . 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 feedback 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.

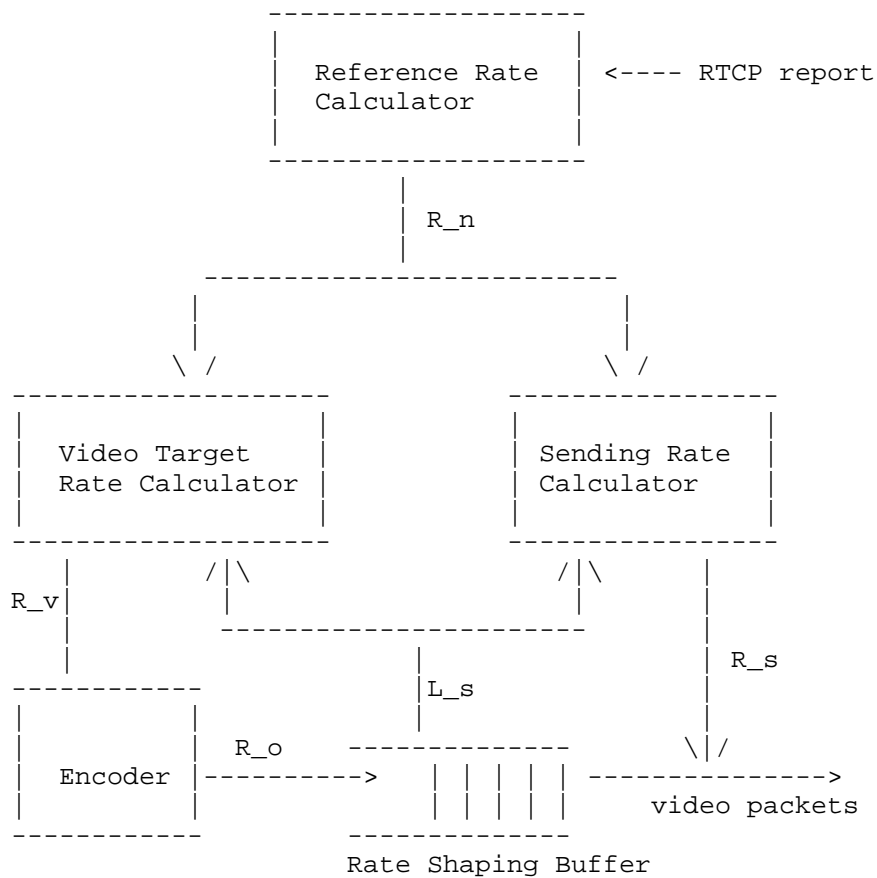


Figure 1 NADA Sender Structure

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:

$$\gamma = \min \left[\gamma_0, \frac{T_{\text{th}}}{\text{RTT}_0 + \delta_0} \right] \quad (3)$$

$$R_n = (1 + \gamma) R_r \quad (4)$$

In (3) and (4), the multiplier γ for rate increase is upper-bounded by a fixed ratio γ_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 δ_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:

$$R_n \leftarrow R_n + \frac{\kappa * \delta_s}{\tau_o^2} * (\theta - (R_n - R_{min}) * x_{hat}) \quad (5)$$

where

$$\theta = w * (R_{max} - R_{min}) * x_{ref}. \quad (6)$$

$$x_{hat} = x_n + \eta * \tau_o * x'_n \quad (7)$$

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

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 η and κ in (5) and (7) can ensure system stability so long as the RTT falls below the upper bound of τ_o . The recommended default value of τ_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:

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

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:

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

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

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

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 τ_v is given by L_s/τ_v . The scaling parameters β_v and β_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 Evaluation 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 Interim, <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 α is a value between 0 and 1. A value of $\alpha=1$ corresponds to performing no smoothing at all.

* calculate marking probability p as:

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

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

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

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, if b-b_tk < b_lo;  
  
  p = p_max*  $\frac{b-b_{tk}-b_{lo}}{b_{hi}-b_{lo}}$ , if b_lo<= b-b_tk <b_hi;  
  
  p = 1, 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.

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