Congestion Control for WebRTC: Standardization Status and Open Issues

Luca De Cicco, Gaetano Carlucci, and Saverio Mascolo

Abstract

The WebRTC initiative has achieved impressive results in terms of the gained industrial interest, the penetration of the technology in end-user devices, and the ever growing community of developers. WebRTC is today supported by major mobile platforms and Internet browsers, allowing potentially billions of users to seamlessly establish real-time communication sessions. Among all the functionalities that must be implemented by WebRTC devices, congestion control is particularly important to ensure that the network operates properly while providing a satisfactory user experience. A working group that is focusing on this issue is the IETF RTP Media Congestion Avoidance Techniques (RMCAT), which aims at defining the requirements and designing congestion control algorithms to be used for the transport of real-time media flows over RTP. This article overviews the status of the standardization efforts that are taking place in the RMCAT working group. We discuss the choices involved in the design of media congestion control, the proposed algorithms, and the issues that are considered still open.

Introduction

Real-time applications, such as video conferencing, gaming, and instant messaging, are becoming widely popular. Despite past standardization efforts in the IETF, such as the Real-Time Protocol (RTP), the industry has been reluctant to employ standard protocols and has resorted to using proprietary protocols and algorithms in their products with the drawback of having services that cannot interoperate. The WebRTC initiative started in 2011 to precisely address the issue of standardizing protocols and APIs to enable real-time services among Web browsers [5]. The achieved results of this initiative in such a short time frame are impressive: WebRTC is already available in popular browsers such as Google Chrome, Firefox, and Opera, and natively in the Android and iOS mobile platforms. Motivated by the huge penetration of this technology in end-user devices, popular applications such as Whatsapp, Facebook Messenger, and Google Duo are today adopting WebRTC to implement real-time services. However, several issues still need to be addressed by the WebRTC community. Among them, a key problem is the reduction of the Internet latency, which is detrimental to the Quality of Experience (QoE) of real-time applications [2]. The way the Internet latency impacts the QoE depends on whether WebRTC is used to transport media, data or both. In this article we focus on WebRTC applications transporting media streams. These applications have challenging requirements since the QoE is not only affected by the latency but also by the audio/video quality, which depends on the used bandwidth and the experienced packet losses. To meet such requirements and avoid network congestion, WebRTC media flows must use a congestion control algorithm. The IETF working group (WG) "RTP Media Congestion Avoidance Techniques" (RMCAT) aims at designing such an algorithm on top of RTP. This article overviews the work that is taking place in the RMCAT WG. We describe the general architecture to implement congestion control in WebRTC, the requirements and design choices, and the used approaches to detect congestion. Then, we summarize the main features of the proposed algorithms and outline the remaining open issues.

Architecture

In this section, we focus on the case of a WebRTC application in which peers want to establish a real-time session involving multiple media streams over RTP. A broader discussion on the architecture of general WebRTC applications can be found in [5]. Figure 1 shows the generic architecture employed by WebRTC media applications. The figure shows that the *i*-th media stream is first compressed by an *encoder* at a bitrate A_i and then fed to a *packetizer* that generates an RTP packet stream. The suggested approach is to multiplex all the RTP packet streams in a single RTP session sent over one transport layer flow to reduce the number of flows to be handled by NATs. Following this approach, the *Media Congestion Control* algorithm computes the total sending rate A for the aggregated RTP session based on a set of metrics measured at the receiver and sent back through RTCP packets. A *rate allocation* module allocates a fraction A_i of the total bitrate A to each media stream. In particular, each media stream is compressed at a target encoding rate A_i such that the sum of all these rates equals the computed sending rate A. The media streams are then multiplexed over a single RTP session that feeds a *sending engine*. This module is responsible for sending the RTP packets to the network at a rate as close as possible to the rate computed by the congestion control algorithm. It

Digital Object Identifier: 10.1109/MCOMSTD.2017.1700014

The authors are with Politecnico di Bari.

is worth noting that the placement of the congestion control algorithm is not specified and could be sender-side, receiver-side, or distributed both at sender and receiver. As an example, Fig. 1 shows the case where the congestion control algorithm is placed only at the sender. The receiver is required to implement a module that measures metrics to be fed back to the congestion control algorithm through RTCP packets. The section "Open Issues" will discuss the implications that such an architectural choice has on system scalability and deployability.

Requirements and Design Choices

The requirements for WebRTC media flows are described in [7] and summarized in Table 1. In a nutshell, based on such requirements, the goal of the congestion control algorithm is to produce a sending rate as close as possible to the available end-to-end bandwidth while maintaining the queue occupancy as low as possible. Additionally, media flows generated by WebRTC applications should fairly share network bandwidth with other concurrent flows. The RMCAT WG considers these QoS-related requirements since this approach has the merit of focusing the discussion on metrics that are not sensitive to application-specific aspects, such as the employed video encoder. The design of an algorithm meeting these requirements is faced with several choices with respect to

1. The transport protocol
2. Congestion detection
3. The actuation mechanism to be employed

Since WebRTC mandates the use of the RTP, several lower-layer transport protocols could be used. However, the transport protocols that are widely supported by NAT traversal protocols (e.g. STUN and TURN) and equipment are TCP and UDP. Between these two protocols, the preferred choice is UDP since real-time flows cannot tolerate inflated latencies due to TCP packet retransmissions. Notice that in the case where UDP cannot be used due to network policy restrictions, the transport layer choice has to fall back to TCP. In this case, end-points have to use the TCP congestion control algorithms provided by the running operating system which might not satisfy real-time requirements. Hence, this article focuses on congestion control algorithms designed at the application layer and transported over RTP/UDP.

Congestion detection can be either *implicit*, when based on end-to-end measurements performed at the end-points, or *explicit*, when congestion is measured directly in network elements by monitoring the router buffers lengths.

The approaches to detect congestion through end-to-end measurements can be divided into two main categories:

1. *Loss-based* algorithms, detecting congestion based on packet loss events
2. *Delay-based* algorithms, detecting congestion based on latency measurements.

Delay-based algorithms are preferred to loss-based algorithms due to two reasons: first, delay-based schemes can detect congestion before packets are lost due to buffer overflows; second, loss-based algorithms cannot control queuing delays since they continuously probe for the network available bandwidth by filling and draining Internet buffers, generating significant delay variations. Notice that explicitly controlling queuing delays is necessary, since excessively large buffers may lead to latencies of the order of seconds [2]. An important issue to be taken into account is to prevent delay-based flows from being starved when competing with loss-based flows in the best-effort Internet [7]. Congestion control algorithms may complement end-to-end measurements with explicit congestion signals sent from network elements to end-points through, for instance, the use of the explicit congestion notification (ECN) mechanism.

Concerning the actuation mechanism, the congestion control algorithm can either compute a congestion window (*window-based* approach) or explicitly compute a sending rate (*rate-based* approach). The use of rate-based mechanisms makes it possible to directly use the rate computed by the congestion control algorithm to drive the media encoders, whereas in the case of window-based algorithms, a proper conversion from a window to a rate should be performed.

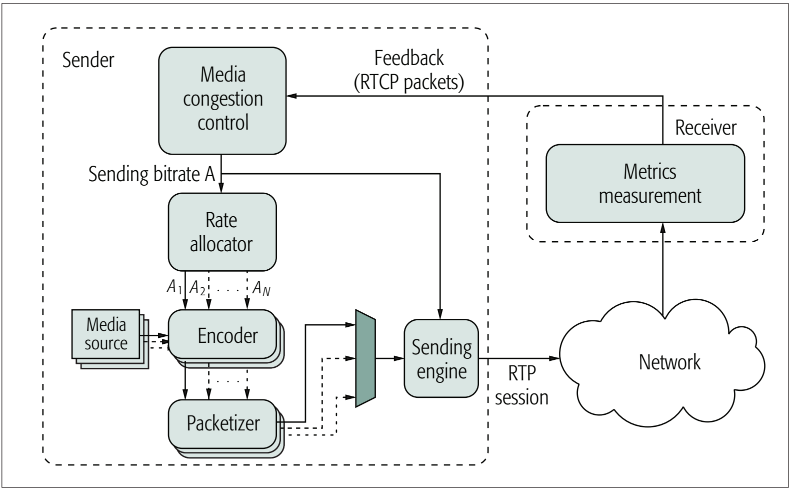

Figure 1. WebRTC media application architecture.

	Requirement
Latency	Possibly lower than 100ms
Packet losses	Should be minimized, FEC mechanism may be employed
Throughput	Should be as high as possible
Burstiness	A smooth sending-rate should be produced
Fairness	Should fairly share the bandwidth with real-time and data flows
Starvation	Media flows should not be starved when competing with TCP flows
Network support	No special network support should be required to operate

Table 1. WEBRTC media flow requirements.

The use of rate-based mechanisms makes it possible to directly use the rate computed by the congestion control algorithm to drive the media encoders, whereas in the case of window-based algorithms, a proper conversion from a window to a rate should be performed.

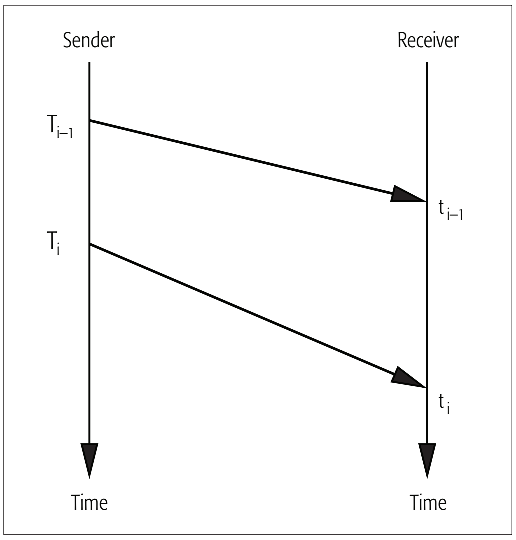

Figure 2. Two consecutive RTP packets are sent at time T_{i-1} and T_i and received at time t_{i-1} and t_i.

Congestion Detection Approaches

This section describes commonly used approaches to detect congestion based on end-to-end delay measurements performed at the clients. These approaches aim at estimating the queuing delay, which is a metric directly correlated to network congestion. In a nutshell, an increasing queuing delay indicates that the bottleneck buffer is being inflated since the filling rate exceeds the link capacity. Intuitively, the congestion control should react to an increase of the estimated queuing delay with a reduction of the sending rate to allow the queue to be drained. These techniques are also used by congestion control algorithms proposed in the RMCAT WG. In the following, we group such approaches based on the measured metric to detect congestion.

Round-Trip Time

Historically, the first metric used by delay-based algorithms (e.g. TCP Vegas and TCP Fast) has been the end-to-end round-trip time (RTT). Once the RTT measurement is done at the sender, the queuing delay is estimated by subtracting the minimum RTT measured since the beginning of the session. Despite its straightforward implementation, this approach has the drawback of measuring the sum of the queuing delay in the forward and backward paths. Thus, the presence of congestion on the backward path may trigger a congestion event even in the absence of congestion on the forward path. Consequently, poor channel utilization might be achieved. It is worth mentioning that the problem of reverse traffic is crucial in the context of video conferencing since video flows are sent in both directions.

One-Way Delay

The instantaneous one-way delay can be measured as the difference between the time instant T_i at which the i-th packet has been sent and the time instant T_i at which the same packet is received (Fig. 2). Thus, the time T_i needs to be stamped in the RTP packet. There are different ways to store this information into the RTP packet that will be discussed later. Then, the one-way queuing delay is estimated by subtracting to the measured one-way delay its minimum value measured during the session. This approach has the advantage of making the queuing delay estimation independent from the backward path state conditions. However, it has been shown that this technique might be affected by the so-called "latecomer effect:" when two flows share the same bottleneck, the flow that arrives later typically starves the first one. This is due to the fact that the last flow arriving at the bottleneck measures a minimum one-way delay that also accounts for the queuing delay of the existing flow. At the same time, the first flow measures an increasing one-way delay due to the presence of the arriving flow. At this point, the first flow yields bandwidth resources to the second flow, which eventually starves the first flow.

One-Way Delay Variation

Given two consecutive RTP packets sent from the sender to the receiver, a one-way delay (OWDV) sample is measured as the difference between two consecutive one-way delay samples, i.e., $T_i - T_i - (t_{i-1} - T_{i-1})$ (Fig. 2). Even in this case, the sending time has to be stored in the RTP packet. Clock synchronization is not required in this case since the offset between sender and receiver clocks is canceled out by the difference between the two samples. The way the OWDV is used to detect congestion differs from the approaches described above since it estimates the queuing delay variation. Basically, a positive measured OWDV indicates that the queuing delay is increasing; conversely, a negative OWDV means that the queuing delay is decreasing. A measured OWDV equal to zero indicates that the bottleneck queuing delay is constant. This can happen in three different conditions:

1. When the queue is empty due to *channel underutilization*, i.e., when the application sending rate is below the link capacity.
2. If the queue is full due to *persistent congestion*, i.e., when the filling rate exceeds the link capacity.
3. When the rate exactly matches the link capacity.

In the third case, the queue stays constant to a value between zero and its maximum size. This is an undesirable situation known as *standing queue*, which steadily delays the incoming traffic. Thus, to guarantee a small queue occupancy while fully utilizing the available bandwidth, the algorithm has to continuously probe for the available bandwidth by increasing its sending rate until a positive queuing delay variation is detected. At this point, the sending rate should be quickly reduced. To summarize, some queuing delay needs to be induced to run a congestion control algorithm based on delay variations.

Proposed Algorithms

This Section describes the design choices, features, standardization and implementation status, and main results of the three congestion control algorithms proposed within the RMCAT WG:

1. Google Congestion Control (GCC) by Google

2. Network Assisted Dynamic Adaptation (NADA) by Cisco
3. Self-Clocked Rate Adaptation for Multimedia (SCReAM) by Ericsson

Algorithms' Designs At A Glance

GCC [3]. The congestion control algorithm throttles the sending rate based on the estimated congestion state. To estimate the state, GCC employs a finite state machine that is driven by a signal obtained by comparing the measured one-way delay variation with a dynamic threshold. In a nutshell, when the bottleneck is estimated as "underused" the sending rate is increased; when it is estimated as "overused" the sending rate is reduced. The use of a dynamic threshold to estimate the congestion state has revealed to be a key design requirement to face with starvation issues when GCC flows share the bottleneck with TCP flows.

NADA [12] computes the sending rate based on both explicit congestion signals provided by network elements (the explicit packet ECN marking ratio) and implicit congestion signals, i.e., the measured one-way queuing delay and loss ratio. The resulting *aggregated congestion signal* is obtained by summing, through appropriate weights, all the contributes defined above. When no assistance is provided by the network, NADA basically operates as a delay-based algorithm taking the estimated one-way delay as the congestion signal. To avoid starvation when competing with loss-based flows, the estimated delay is first passed to a non-linear function which artificially decreases the estimated delay when it exceeds a threshold.

SCReAM [4] estimates the one-way queuing delay as the difference between the measured one-way delay and the minimum observed one-way delay, which is defined as the base delay. The algorithm employs a congestion window (*cwnd*) to limit the number of inflight bytes. The *cwnd* can increase only if the value of the estimated one-way queuing delay is below a given threshold (typically 50–100 ms), otherwise it decreases the *cwnd* proportionally to the difference between the threshold and the one-way queuing delay. SCReAM dynamically adjusts the target for the queuing delay to cope with starvation issues when competing with loss-based flows. Finally, a module, the *Media Rate Control*, is used to compute the sending rate based on the congestion window.

Algorithms' Features

Table 2 summarizes the main features of the proposed algorithms comprising the metrics employed to infer congestion, the employed architecture, and the implementation status.

Concerning the metrics employed to detect congestion, NADA and SCReAM employ the one-way delay, whereas GCC employs the one-way delay variation. Besides the delay-based congestion control mechanism, all the proposed algorithms also comprise a mechanism reacting to losses. To this purpose, the fraction of lost packets is used (see RTP RFC for details). It is important to stress that the loss-based component is employed as a fall-back when the delay-based algorithm cannot prevent losses due to buffer overflow (for instance when the bottleneck buffer is small).

Feature	GCC	NADA	SCReAM
Metrics	One-way delay variation, loss ratio	One-way delay, loss ratio	One-way delay, loss ratio
Architecture	Sender-side or hybrid	Sender-side	Sender-side
Actuation mechanism	Rate-based	Rate-based	Window-based
Network support	None	ECN, PCN	ECN
Implementation status	Google Chrome	Ns-2 and Ns-3 simulators	OpenWebRTC and simulator
Codec interaction	VP8 and VP9	Simulated encoder	OpenH264 and VP9

Table 2. RMCAT candidate algorithms features.

All the proposed algorithms follow an architectural design approach similar to the approach shown in Fig. 1. Specifically, NADA and SCReAM implement the control logic at the sender, whereas the receiver only performs measurements that are reported through RTCP feedbacks. GCC was first conceived as a hybrid sender/receiver side algorithm; recently it has been re-designed to support a sender-side only architecture. Concerning the actuation mechanism, SCReAM is the only window-based algorithm proposed in RMCAT. On the other hand, both GCC and NADA explicitly compute a sending rate that is then used to drive the encoders as described earlier. Even though the proposed algorithms do not require network cooperation in order to operate, SCReAM supports ECN, wheres NADA supports both ECN and the pre-congestion notification (PCN) as complementary congestion signals. The support of explicit feedback from network elements has also been considered by GCC authors, but at the time of this writing it has not yet been specified.

Standardization and Implementation Status

Regarding the standardization status, SCReAM has passed the final review of the working group last call (WGLC), whereas NADA is about to enter this phase shortly. As such, both SCReAM and NADA are expected to be submitted to the Internet Engineering Steering Group (IESG) for experimental publication in the forthcoming months. Concerning GCC, the draft has to address some issues in the loss-based component of the algorithm before entering the WGLC phase.

Regarding the algorithms' implementation status, the situation is quite heterogeneous. NADA has been implemented in the Ns-2[1] and Ns-3 simulators, whereas SCReAM and GCC have real implementations. SCReAM has been implemented in OpenWebRTC,[2] an open-source project that can be used to build native WebRTC applications. However, the maintained version of the algorithm is only available in the form of a simulator.[3] GCC is included in the official release of Google Chrome and it is also being used by Google Hangouts and the Google Duo mobile application. As a matter of fact, GCC is the only proposed congestion control algorithm that has been integrated into a widely-used browser. As such, GCC can be easily tested in real network

[1] This is the only implementation that the authors have publicly released.

[2] https://www.openwebrtc.org/

[3] https://github.com/EricssonResearch/scream.

A WebRTC application may open multiple media flows generated by different media sources that could share the same bottleneck. In such a situation, a mechanism to share states across the congestion control algorithms transporting such flows can be used to allow taking better decisions in the allocation of bandwidth resources.

environments by starting a video conferencing session with the official Google Chrome browser and using AppRTC,[4] a popular WebRTC demo application developed by Google.

Results

Due to the different implementation status, it is difficult to provide a quantitative comparison between the performance of the proposed algorithms. As such, in the following, we qualitatively compare the results obtained independently by authors in the common network scenarios defined by the RMCAT WG [9]. Concerning NADA and SCReAM, the following discussion is based on the simulative results presented by the algorithms' authors in the RMCAT WG meetings.[5,6] Regarding GCC results, we refer to the experimental evaluation reported in [3] that was carried out with Google Chrome browsers establishing real WebRTC video conferencing sessions in an emulated WAN network.

The test case **"Variable Available Capacity"** aims at assessing the responsiveness of the algorithm to step-like increases and decreases of the bottleneck link capacity in the single-flow case and when multiple media flows share the bottleneck. In this scenario, NADA adapts the sending rate to the link capacity variations responsively when an ideal codec is used, whereas it exhibits remarkable sending rate oscillations and inflated latencies when a trace-based codec is used in Ns-3 simulations. SCReAM nicely adapts the sending rate in the single flow scenario when the RTT of the path is small, but provokes a standing queue when the RTT is larger. This issue is due to the mechanism implemented by SCReAM to cope with TCP long-lived flows. In this scenario, GCC is able to adapt the sending rate and match the channel capacity containing the queuing delays.

The test case **"Competing Media Flows"** tests inter-protocol fairness requirements by considering three concurrent media flows over a link with constant capacity. NADA provides fast rate convergence among concurrent flows when an ideal codec is used, but it exhibits remarkable rate oscillations when a trace-based codec is used. SCReAM and GCC sending rates convergence time is higher but they provide lower queuing delays with respect to NADA. All the candidate algorithms prevent packet losses during the experiments in this test case.

Finally, the test case **"Media Flow Competing with a Long TCP Flow"** assesses inter-protocol fairness when one media flow shares a constant capacity bottleneck link with one long-lived TCP flow. GCC and NADA are able to avoid starvation against the TCP flow since they make their control action more aggressive in such situations. NADA convergence is faster compared to GCC, but NADA and TCP rates remarkably oscillate when a trace-based codec is used. Concerning SCReAM, results show that TCP starves the media flow.

To summarize, the results obtained by NADA show that the algorithm still needs to be tuned to work with real codecs, whereas SCReAM fails to avoid starvation when used concurrently with TCP flows.

Open Issues

In this section, we briefly provide a discussion of complementary issues related to the definition of a congestion control algorithm for media transport.

Interactions with Concurrent Data or Media Flows

A WebRTC application may open multiple media flows generated by different media sources that could share the same bottleneck (Fig. 1). In such a situation, a mechanism to share states across the congestion control algorithms transporting such flows can be used to allow taking better decisions in the allocation of bandwidth resources. The way the congestion state should be shared among media flows is specified in [11] which has passed the WGLC and has been submitted to the IESG for experimental publication.

Besides media transport, WebRTC also provides reliable transport of data through data channels [5]. Data channels allow prioritization among data streams over the Stream Control Transmission Protocol (SCTP) which uses its own congestion control. Typically, in order to minimize the number of NAT bindings and consequently the risk of NAT traversal failure, media and data traffic are multiplexed over a single lower layer protocol. At the time of this writing, the discussion of the interaction between data and media congestion control mechanisms is still at an early stage.

RTP/RTCP Extensions

The RTP/RTCP standard is considering new extensions designed for enhancing media congestion control functionalities, such as absolute timestamps[7] or relative transmission time offsets in RTP header extensions [10]. These are two techniques used to stamp the departure time of an RTP packet in their header. This is important information that is required to perform delay measurements. Since the RTP standard does not define a transmission timestamp, the "relative transmission time offsets extension gives information on the relationship between the time spacing of the transmission time and the RTP timestamp used for media synchronization. Recently, Google uncovered that using relative send time offsets could be problematic when packets are relayed through middleboxes. To prevent middlebox interference, Google has proposed the *absolute send time* extension, which consists of adding four bytes to RTP packets to add the timestamp of the departure time from the system that puts this packet on the wire.

Among other RTP/RTCP extensions that are related to congestion control, we mention:

1. The *circuit-breaker* [6] algorithm, which is used to stop a media session when congestion control is not operating efficiently.
2. REMB [1], which is an RTCP message for signaling the receiver estimated maximum bitrate that should be employed by a congestion control algorithm.

Feedback Message Format

Recently, the RMCAT WG has been discussing the definition of a generic RTCP feedback message format that should be used by candidate congestion control algorithms [8]. In such a way, sender-side algorithms would only require

[4] https://appr.tc

[5] "SCReAM update and test case results," IETF-96 RMCAT meeting, https://www.ietf.org/proceedings/96/slides/slides-96-rmcat-0.pdf

[6] Update on NADA Performance Evaluation, IETF-96 RMCAT meeting, https://www.ietf.org/proceedings/96/slides/slides-96-rmcat-3.pdf

[7] https://webrtc.org/experiments/rtp-hdrext/abs-send-time/

the receiver to measure a set of defined metrics to be reported by a common feedback message. This approach has interesting implementation advantages. In fact, from the perspective of congestion control, the required functionalities to be implemented would be greatly simplified. In particular, the receiver could be made agnostic to the employed congestion control algorithm at the sender, which is beneficial in the case of multi-party scenarios where multiple receivers are handled by a multi-point control unit (MCU).[8]

Common Experimental Testbed Platform

The definition of a common testbed for the evaluation of different congestion control algorithms is at an early stage. Such a common testbed is indeed the missing piece to fairly compare the proposed algorithms in the common scenarios defined in the RMCAT WG.

An interesting initiative in this direction is the "Pantheon of Congestion Control" (PCC)[9] which has been proposed by Standford University in the context of the IETF Internet Congestion Control Research Group (ICCRG) WG. PCC has been proposed to compare the performance of congestion control algorithms in terms of self-inflicted queuing delays and obtained throughput. Among the algorithms proposed in RMCAT, at the moment PCC only includes GCC and SCReAM. GCC is tested by employing a real WebRTC session with a real video using Google Chrome end-points, whereas SCReAM is tested with an ideal encoder. We believe that building on the ideas of PCC, the RMCAT community could develop within a reasonable amount of work a common platform to experimentally evaluate all the proposed algorithms.

GCC is tested by employing a real WebRTC session with a real video using Google Chrome end-points, whereas SCReAM is tested with an ideal encoder. We believe that building on the ideas of PCC, the RMCAT community could develop with a reasonable amount of work a common platform to experimentally evaluate all the proposed algorithms.

Summary

The IETF RTP Media Congestion Avoidance Techniques (RMCAT) Working Group will specify one or more congestion control algorithms for the transport of real-time media over RTP. This feature has to be implemented in end-points to guarantee both the safe deployment of WebRTC-based applications over the Internet and to provide the highest possible media quality to the users. Three algorithms have been proposed in the WG and they have been implemented and tested in different environments. This heterogeneity does not allow a fair comparison among the solutions and, for this reason, some proposals for the specification of a common experimental testbed platform have been made. Moreover, the interaction between the media and data channel congestion control algorithms also needs to be specified to allow for fair sharing of network resources and flow prioritization.

References

[1] H. Alvestrand, "RTCP Message for Receiver Estimated Maximum Bitrate," Internet-Draft draft-alvestrand-rmcat-remb-03 (work in progress), Oct. 2013.

[2] B. Briscoe *et al.*, "Reducing Internet Latency: A Survey of Techniques and Their Merits," *IEEE Commun. Surveys Tutorials*, vol. 18, no. 3, 2016, pp. 2149–96.

[3] G. Carlucci *et al.*, "Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC)," *Proc. ACM Multimedia Systems Conf.*, Klagenfurt, Austria, May 2016.

[4] I. Johansson, "Self-clocked Rate Adaptation for Conversational Video in LTE," *Proc. 2014 ACM SIGCOMM Wksp. Capacity Sharing Workshop*, Chicago, USA, Aug. 2014, pp. 51–56.

[5] S. Loreto and S. P. Romano, "Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts," *IEEE Internet Computing*, vol. 16, no. 5, Sept. 2012, pp. 68–73.

[6] C. Perkins and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions," RFC 8083, RFC Editor, Mar. 2017.

[7] J Randell and Z. Sarker, "Congestion Control Requirements for RMCAT," Internet-Draft draft-ietf-rmcat-cc-requirements-09 (work in progress), Dec. 2014.

[8] Z. Sarker*et al.*, "RTP Control Protocol (RTCP) Feedback for Congestion Control," Internet-Draft draft-dt-rmcat-feedback-message-01 (work in progress), Oct. 2016.

[9] Z. Sarker *et al.*, "Test Cases for Evaluating RMCAT Proposals," Internet-Draft draft-ietf-rmcat-eval-test-04 (work in progress), Oct. 2016.

[10] D. Singer and H. Desineni, "Transmission Time Offsets in RTP Streams," RFC 5450, RFC Editor, Mar. 2009.

[11] M. Welzl, S. Islam, and S. Gjessing, "Coupled Congestion Control for RTP Media," Internet-Draft draft-ietf-rmcat-coupled-cc-06 (work in progress), Mar. 2017.

[12] X. Zhu *et al.*, "NADA: A Unified Congestion Control Scheme for Real-Time Media," Internet-Draft draft-ietf-rmcat-nada-04 (work in progress), Mar. 2017.

Biographies

Luca De Cicco (luca.decicco@poliba.it) received both the laurea degree and the Ph.D. from Politecnico di Bari, Italy, where he is currently an assistant professor. He has held visiting positions at the University of New Mexico (NM, USA), the Centrale SUPELEC (Gif-sur-Yvette, France), and at the Laboratory of Information, Networking and Communication Sciences-LINCS (Paris). His main interests are the modeling and design of congestion control algorithms for multimedia transport, adaptive video streaming, SDN, and server overload control.

Gaetano Carlucci (gaetano.carlucci@poliba.it) received both the M.Sc. degree and the Ph.D. degree from Politecnico di Bari, Italy. He held a visiting position at the Video and Content Platforms Research and Advanced Development Group at CISCO in Boston, USA, in 2014. His main interests focus on the modeling and design of control algorithms and transport protocols for low delay communication over the Internet.

Saverio Mascolo (saverio.mascolo@poliba.it) received both the laurea degree and the Ph.D. from Politecnico di Bari, Italy, where he is currently a full professor and Chair of the Department of Electrical Engineering and Computer Science. He has worked on nonlinear control, synchronization of chaotic systems, modelling and control of data networks, congestion control, adaptive video streaming, content delivery networks, software-defined networks, and server overload control. Currently he is an associate editor of *IEEE/ACM Transactions on Networking* and Elsevier *Computer Networks Journal*.

[8] https://janus.conf.meetecho.com/

[9] https://github.com/StanfordLPNG/pantheon