Congestion Control for Web Real-time Communication

Gaetano Carlucci, Luca De Cicco, *Member, IEEE*, Stefan Holmer, and Saverio Mascolo, *Senior Member, IEEE*

***Abstract*—Applications requiring real-time communication (RTC) between Internet peers are ever increasing. Real-time communication requires not only congestion control but also minimization of queuing delays to provide interactivity. It is known that the well-established TCP congestion control is not suitable for real-time communication due to its retransmissions and in-order delivery mechanisms which induce significant latency. In this paper we propose a novel congestion control algorithm for RTC which is based on the main idea of estimating – using a Kalman filter – the end-to-end one-way delay variation which is experienced by packets traveling from a sender to a destination. This estimate is compared with a dynamic threshold and drives the dynamics of a controller located at the receiver which aims at maintaining queuing delays low, while a loss-based controller located at the sender acts when losses are detected. The proposed congestion control algorithm has been adopted by Google Chrome. Extensive experimental evaluations have shown that the algorithm contain queuing delays while providing intra and inter protocol fairness along with full link utilization.**

I. Introduction

VIDEO constitutes the largest part of the Internet traffic according to recent measurement studies [1], [2]. Although video streaming is the primary driver of this growth, applications generating audio/video flows for establishing end-to-end real-time communication (RTC) are also getting very popular. This is mainly due to the ever increasing diffusion of hand-held devices (f.i., smartphones, tablets) which capture, encode, and send real-time video flows through mobile connections. Besides traditional video conferencing and telepresence systems, new mobile applications, such as Periscope, Meerkat or Facebook live, allowing videos captured by smartphones to be streamed in real-time, are getting momentum.

Even though the Internet has undergone significant changes in its upper layers and today is employed as a platform for delivering video at a massive scale, the majority of Internet traffic is still mostly delivered through the Transmission Control Protocol (TCP). As a matter of facts, the loss-based congestion control employed by the TCP has proven to be suitable for both elastic data transfer (web browsing, file transfer) and traffic with weak real-time characteristics such as the one generated by video streaming systems which today deliver videos over HTTP/TCP. However, it is well-known that the TCP is not suitable to deliver traffic with hard real-time constraints (*delay-sensitive* traffic), such as the one generated by video conferencing applications. In fact, compared to bulk data delivery, which essentially requires the minimization of flow completion times [3], [4], the Quality of Experience (QoE) of real-time multimedia applications is not only affected by the goodput but also by the connection latency that must be kept as low as possible [5]. It is known that the well-established TCP congestion control is not suitable for real-time communication due to its retransmissions and in-order delivery mechanisms which induce significant latency. To the best of our knowledge, except in the case of VoIP traffic [6], [7], the delivery of real-time video content over TCP – which indeed entails much higher data rates – has never been addressed in the literature nor proven to be successful in real applications. Consequently, despite several standardization efforts – the most notable being DCCP [8], real-time video applications employ UDP sockets managed by ad-hoc congestion control algorithms implemented at the application layer (see f.i., [9] and [10]). The obvious drawback of resorting to this practice is that different applications cannot inter-operate which hinders mass adoption of RTC applications. A joint W3C and IETF initiative called WebRTC has been established to address this issue. In particular, the WebRTC initiative aims at standardizing an interoperable and efficient framework for real-time communication using Web browsers over the Real Time Protocol (RTP) [11]. Launched only a few years ago, today the WebRTC initiative allows more than two billion of users to communicate in real-time through Web browsers[1]. Another related IETF working group, the *RTP Media Congestion Avoidance Techniques*[2] (RMCAT), has been established for standardizing inter-operable congestion control algorithms for RTC.

This paper significantly extends our previews work [12] and presents the *Google Congestion Control* (GCC) which is an algorithm fully compliant with the WebRTC framework. The algorithm has been designed to work with RTP/RTCP protocols and is based on the idea of using delay variations to infer congestion. First, we propose to use a Kalman filter to estimate the one-way delay variation at the application layer. Then, we show that the estimated delay variation cannot be compared to a static threshold to detect congestion, and we propose a simple and yet effective control law to dynamically adapt the threshold.

In our previous work [12] we experimentally evaluated GCC in the scenarios described in the IETF RMCAT working

Gaetano Carlucci is Post-Doc at Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Italy. gaetano.carlucci@poliba.it

Luca De Cicco is Assistant Professor at Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Italy. luca.decicco@poliba.it

Stefan Holmer is Research Engineer at Google Inc., Sweden. holmer@google.com

Saverio Mascolo is Full Professor at Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Italy. mascolo@poliba.it

[1] http://iswebrtcreadyyet.com/

[2] http://datatracker.ietf.org/wg/rmcat/

group [13]. This choice was made to allow a comparison with other congestion control algorithms proposed in the RMCAT WG on a common set of scenarios. In this paper, we have further investigated the behavior of GCC on a broad set of network conditions and scenarios. In particular, we have evaluated the sensitivity of the algorithm with respect to different queue sizes, bottleneck capacity, and number of concurrent flows. We believe that the obtained results give a more complete picture of the algorithm performance.

The algorithm we propose in this paper is today adopted by Google Chrome, which is the most used browser in the Internet[3]. To the best of our knowledge, GCC is the only proposed congestion control algorithm for WebRTC that has been deployed on the Internet. Moreover, Google has recently announced that the new video conferencing application for Android named *Google Duo* will also employ GCC. Finally, we point out that the experimental results presented in this work can be reproduced. In fact, (i) the code of the proposed algorithm is made available in the Google Chromium GIT repository [14] and (ii) details on how to set up the testbed employed in this paper are publicly released [15].

The rest of this paper is organized as follows. Section II reviews the relevant literature on congestion control for delay-sensitive flows. Section III describes the proposed algorithm. In Section IV control parameters are tuned. Section V presents the experimental testbed and the employed metrics. Section VI illustrates the experimental results and Section VII concludes the paper.

II. Related work

Traditional loss-based TCP is not suitable for real-time communication traffic since its congestion control continuously probes for network available bandwidth introducing periodic cycles during which network queues are first filled and then drained. These queue oscillations induce a time-varying stochastic delay component that adds to the propagation time and makes delay-sensitive communications problematic. Two complementary approaches can be employed to tackle this issue: end-to-end, placing the control in the end-points, and active queue management (AQM), addressing the problem in the routers.

The idea that network delay can be correlated to network congestion has been proposed in the seminal paper [16]. However, since then several issues related to delay measurements in delay-based algorithms have been considered [17], especially in the case of wireless environments [18] and when the bottleneck is shared with loss-based flows [19], [20]. In the following, we provide a review of the related work clustering proposed end-to-end congestion control algorithms based on the metric used to infer congestion and complementary solutions which employ AQM to control bottleneck queuing delays in the network.

A. The use of round trip time to infer congestion

The first efforts aiming at reducing queuing delay were set in the TCP congestion control research domain and, consequently, many algorithms for real-time traffic are rooted in this literature. The first congestion control algorithm specifically designed to contain the end-to-end latency is employed in the seminal work by Jain which dates back to 1989 [16]. Since then, several delay-based TCP congestion control variants have been proposed, such as TCP Vegas [21] and TCP FAST [22] which use RTT measurements to infer congestion. It has been shown that when the RTT is used as a congestion metric a low channel utilization may be obtained in the presence of reverse traffic or when competing with loss-based flows [19]. 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.

B. One-way delay to infer congestion

Another class of algorithms advocates the use of one-way delay measurements to rule out the sensitivity to the reverse-path congestion. Examples are LEDBAT (over UDP) [23] and TCP Santa Cruz [24]. In particular, LEDBAT [23] increases its congestion window at a rate that is proportional to the distance between the measured one-way delay and a fixed delay target. It has been shown that LEDBAT is affected by the so-called *"latecomer effect"*: when two flows share the same bottleneck the second flow typically starves the first one [25].

C. The use of delay-gradient to infer congestion

The idea of employing RTT gradient to infer congestion has been recently used to overcome the aforementioned *"latecomer effect"*. Some examples are CDG [26] and Verus [27]. CDG [26] has been designed to provide fair coexistence with loss-based flows and low end-to-end delay. Verus [27] has been specifically designed for cellular networks where sudden link capacity variations make the congestion control design challenging. Recently, it has been shown that accurate delay gradient measurement is achievable in data center networks by employing NIC hardware timestamps [28].

D. Other approaches

Among recently proposed congestion control algorithms, which do not infer congestion by measuring network delays, we cite Sprout [29], and Remy [30]. Sprout [29] takes a stochastic approach which aims at containing delays while maximizing the throughput. Remy [30] is a framework to generate congestion control algorithms. By defining a utility function based on users requirements, Remy employs apriori knowledge of the network to train a machine that learns congestion control schemes.

E. Design for RTP/RTCP

This paper aims at designing a congestion control algorithm for real-time communication among Web browsers. The algorithm will conform to the WebRTC W3C and IETF joint initiative [11]. Three end-to-end algorithms have been proposed within the IETF RMCAT working group: (i) the Network Assisted Dynamic Adaptation (NADA) [31] by

[3]http://www.w3schools.com/browsers/default.asp

Cisco, is a loss/delay-based algorithm that relies on "one-way delay" measurements; (ii) the Self-Clocked Rate Adaptation for Multimedia (SCREAM) [32] by Ericsson which inherits some ideas from LEDBAT; (iii) Google Congestion Control (GCC) [33] is proposed in this paper. Further details on the standardization status of such algorithms are available in [34].

F. AQM algorithms to reduce the queuing delays

Queuing delays can also be reduced with appropriate tuning of network buffers size [35], [36], [37] or with AQM algorithms which control the router buffers by dropping the packets or marking them if ECN is used [38]. Despite the fact that many AQM algorithms have been proposed in the past, their adoption has been held back due to two main issues [39]: (i) they aim at controlling the average queue length instead of queuing delay and (ii) an ad-hoc configuration of their parameters has to be made. These issues, along with the bufferbloat phenomenon [40], have motivated the study of new AQM algorithms, such as CoDel [39] and PIE [41], that do not require parameter tuning and that explicitly control the queuing delay instead of the queue length.

In the complementary paper [42] we have carried out an experimental investigation studying the interaction between GCC and delay-based AQM schemes. Results show that if only GCC flows run through the bottleneck, GCC is able to contain the queuing delay with zero losses in the case of a drop-tail queue. On the other hand, PIE and CoDel provide roughly the same queuing delay of drop-tail but with the drawback of introducing packet losses. This is because CoDel (or PIE) reacts to the delay inflation before GCC does, with the consequence of inducing losses on the video flow. Moreover, we have shown that flow schedulers such as Stochastic Fair Queuing (SFQ) offer a better solution compared to AQMs since they provide flow isolation. In particular, GCC obtained the best results in terms of queuing delay and packet losses when used with SFQ.

III. The Congestion Control Algorithm (GCC)

GCC is designed for real-time flows. It must provide (i) low queuing in the absence of concurrent heterogeneous traffic and (ii) a reasonable share of bandwidth when competing with homogeneous or heterogeneous flows [43].

In order to satisfy both the requirements two cooperating congestion control algorithms are designed as shown in Figure 1. A *delay-based* controller located at the receiver aims at maintaining queuing delays low, whereas a *loss-based* controller located at the sender acts when losses are detected.

The sender employs a UDP socket to send RTP packets and receive RTCP feedback reports from the receiver. In particular, the *delay-based* controller computes the rate A_r which is fed back to the sender; the *loss-based* controller computes the rate A_s. The target sending bitrate A is set as the minimum of (A_r, A_s). The block *sending engine* encodes the raw video captured from a video source at a bitrate matching A and sends the encoded video through a UDP socket.

In the following, we provide a description of the algorithm. The source code of GCC is available in the WebRTC repository[4] of the Chromium web browser.

A. The delay-based controller

Design Rationale

Ideally, the congestion control algorithm should provide full link utilization while keeping zero queuing at steady state. A direct measurement of the queue length is not available at the end-points. Thus, the queue length must be estimated using the one-way delay or the RTT measurements which are however affected by several issues discussed in Section II, i.e. reverse-path congestion, latecomer phenomenon, etc.

In order to eliminate such issues, we propose to measure one-way delay variations to detect congestion. The architecture of the proposed delay-based controller is detailed in Figure 1.

In a nutshell, we propose to design a delay-based controller that in response to an increased queuing delay decreases the sending rate, whereas when the queue is drained, it increases the sending rate. To the purpose, we need: (i) a component producing estimates $m(t)$ of the one-way queuing delay variations based on end-to-end measurements (the *Arrival filter* block in Figure 1); (ii) a component that, based on such estimates, detects the state s of the network (the *Overuse detector* block in Figure 1); (iii) a *Rate controller* that computes the rate A_r based on the detected network state s.

In the following, we present the design of these three components.

The delay variation estimation

Definition 1: The *one-way queuing delay gradient* is the derivative of the queuing delay $T_q(t)$.

A well-known fluid-flow model of the queuing delay is $T_q(t) = q(t)/C$, where C is the bottleneck link capacity and $q(t)$ is the queue length measured in bits [44]. Thus, the queuing delay gradient $\dot{T}_q(t)$ is equal to:

$$\dot{T}_q(t) = \frac{\dot{q}(t)}{C} \tag{1}$$

The derivative of the queue length can be modeled as follows [45]:

$$\dot{q}(t) = \begin{cases} r(t) - C & 0 \leq q(t) \leq q_M \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $r(t)$ is the queue filling rate measured in bit per second and q_M is the queue size. $\dot{T}_q$ can be used as a congestion signal since, when $\dot{T}_q(t) > 0$ the queue is inflating, conversely when $\dot{T}_q(t) < 0$ the queue is deflating. In all cases, the higher the value of $|\dot{T}_q(t)|$ the higher the rate the queue is filled or drained.

The case $\dot{T}_q(t) = 0$ needs to be analyzed separately. $\dot{T}_q(t) = 0$ implies $\dot{q}(t) = 0$, i.e. the queue length $q(t)$ stays constant. This can happen in three different conditions: (i) when the filling rate $r(t)$ is below the link capacity C, i.e. in the case

[4]https://chromium.googlesource.com/external/webrtc/+/master/webrtc/

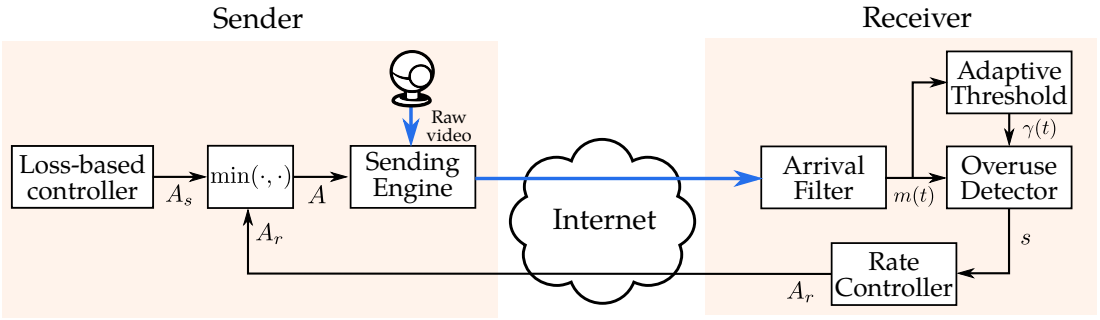


Figure 1. Google Congestion Control architecture

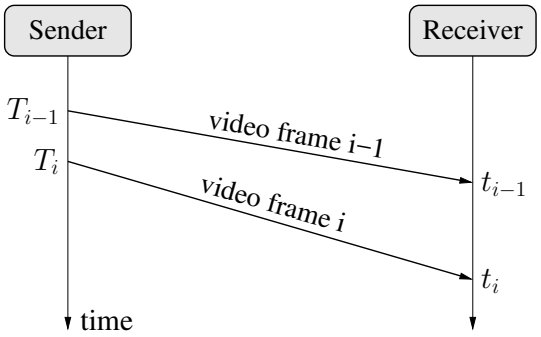


Figure 2. Measurement of the one-way delay variation d_m

of *channel underutilization,* the queue eventually gets empty; (ii) when the filling rate $r(t)$ exceeds the link capacity C, i.e. when *persistent congestion* occurs, the queue keeps equal to q_M; (iii) when the input rate $r(t)$ is exactly equal to C. In this last case, the queue stays at a constant value $\overline{q} \in [0, q_M]$. The case of *standing queue* [39], i.e. $\overline{q} > 0$, is regarded as undesirable since it steadily delays the traffic.

As mentioned above, the proposed algorithm aims at keeping the queue as small as possible without underutilizing the link. To achieve this goal the algorithm has to probe for the available bandwidth by increasing its sending rate until a positive queuing delay variation is detected. At this point, the sending rate must be reduced. Thus, some queuing delay needs to be induced to run a congestion control algorithm based on delay variations.

Estimation. Measuring one-way delays in computer networks is a key issue in several applications ranging from synchronization of distributed nodes [46] to delay-based congestion control algorithms [17]. Despite the fact that the use of delay to infer congestion has been already debated in the past [47], recently several studies are advocating the use of such a metric to drive congestion control algorithms [26], [28].

We propose to employ one-way delay variation (OWDV) end-to-end measurements $d_m(t_i)$ to estimate the one-way queuing delay variation $m(t)$. End-points can measure the OWDV $d_m(t_i)$ as follows (see Figure 2):

$$d_m(t_i) = (t_i - T_i) - (t_{i-1} - T_{i-1}) = (t_i - t_{i-1}) - (T_i - T_{i-1}) \tag{3}$$

where T_i is the time (stamped in the packet) when the first packet of the i-th video frame has been sent[5] and t_i is the time when the last packet that forms the i-th video frame has been received. It should be noted that (3) does not require sender and receiver clock synchronization.

The model $d(t_i)$ of the measured one-way delay variation $d_m(t_i)$ is given by the sum of three components: (i) the transmission time variation, given by the ratio between the variation of the size of two consecutive video frames $\Delta L(t_i)$ and the bottleneck link capacity $C(t_i)$, (ii) the one-way queuing delay variation $m(t_i)$, and (iii) the measurement noise $n(t_i)$:

$$d(t_i) = \frac{\Delta L(t_i)}{C(t_i)} + m(t_i) + n(t_i).$$

The goal here is to extract $m(t_i)$ from $d(t_i)$. Specifically, we need a tool that is able to both filter out the noise $n(t)$ and eliminate the term due to the transmission delay variation. The use of a noise filter on $d_m(t_i)$ would only be able to filter out the component $n(t_i)$ and not the contribution due to the transmission delay variation. A suitable and robust tool to solve this problem is the Kalman Filter [48]. We employ a Kalman filter to estimate the state $\boldsymbol{\theta}(t_i)$ of the system, based on its model and on the noisy measurements $d_m(t_i)$ (3). The system state vector is defined as follows [6]:

$$\boldsymbol{\theta}(t_i) = \begin{bmatrix} \frac{1}{C(t_i)} \\ m(t_i) \end{bmatrix} \tag{4}$$

The system model is given by:

$$\boldsymbol{\theta}(t_{i+1}) = \boldsymbol{\theta}(t_i) + \boldsymbol{w}(t_i) \tag{5}$$

The state noise $\boldsymbol{w}(t_i)$ is modeled as a stationary Gaussian process with zero mean and variance $\boldsymbol{Q}(t_i) = E[\boldsymbol{w}(t_i) \cdot \boldsymbol{w}^T(t_i)]$. Similarly to $\boldsymbol{w}(t_i)$, the measurement noise $n(t_i)$ – which takes into account the network jitter – is also considered as a stationary Gaussian process with zero mean and variance $\sigma_n^2(t_i) = E[n(t_i)^2]$. Both the state and measurement noise

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

[6] Throughout the paper vectors are represented with boldface fonts to differentiate them from scalars.

variance are important parameters which are required to be tuned appropriately.

At each step the *innovation* or *residual* $z(t_i)$ is computed as:

$$z(t_i) = d_m(t_i) - \boldsymbol{H}(t_i) \cdot \boldsymbol{\theta}(t_i) \tag{6}$$

where $\boldsymbol{H}(t_i) = \begin{bmatrix} \Delta L(t_i) & 1 \end{bmatrix}$. The innovation $z(t_i)$ is multiplied by the Kalman gain $\boldsymbol{K}(t_i)$ which provides the correction to the estimate and is dynamically updated according to the process and the measurement noise variance:

$$\boldsymbol{K}(t_i) = \frac{(\boldsymbol{P}(t_{i-1}) + \boldsymbol{Q}(t_i))\boldsymbol{H}^T(t_i)}{\boldsymbol{H}(t_i)(\boldsymbol{P}(t_{i-1}) + \boldsymbol{Q}(t_i))\boldsymbol{H}^T(t_i) + \sigma_n^2(t_i)} \tag{7}$$

where $\boldsymbol{P}(t_i)$ is the system error variance recursively computed as:

$$\boldsymbol{P}(t_i) = (\boldsymbol{I} - \boldsymbol{K}(t_i)\boldsymbol{H}(t_i))(\boldsymbol{P}(t_{i-1}) + \boldsymbol{Q}(t_i)) \tag{8}$$

In our case, the Kalman gain is made of two components $\boldsymbol{K}(t_i) = \begin{bmatrix} k_c(t_i) & k_m(t_i) \end{bmatrix}^T$; $k_m(t_i)$ provides the correction to the one-way delay variation $m(t_i)$ as follows:

$$m(t_i) = (1 - k_m(t_i)) \cdot m(t_{i-1}) + k_m(t_i) \cdot (d_m(t_i) - \frac{\Delta L(t_i)}{C(t_{i-1})}). \tag{9}$$

Observe that (9) turns out to be equivalent to an adaptive EWMA filter which takes as input the measured OWDV $d_m(t_i)$ from which it is subtracted the estimated transmission delay variation $\Delta L(t_i)/C(t_{i-1})$. This equation shows that the Kalman filter is able to both adaptively filter the noise $n(t_i)$ and eliminate the contribution of the transmission time variation from the measurements $d_m(t_i)$.

Regarding the tuning of the state noise covariance $\boldsymbol{Q}$, we have run several experiments on real networks and chose the setting that turned out to best balance between algorithm responsiveness in detecting congestion and filtering noise to avoid false positives. The following $\boldsymbol{Q}$ has been obtained:

$$\boldsymbol{Q} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-3} \end{bmatrix}$$

A physical interpretation of such a setting can be gathered by considering the dynamics of the state components. Since the bottleneck capacity C is typically unknown but constant,[7] the variance of such a component (q_{11}) turns out to be very small compared to the variance of the noise affecting m (q_{22}).

Regarding the measurement noise variance $\sigma_n^2(t_i)$, we use an exponential moving average filter of the residual (6):

$$\sigma_n^2(t_i) = \beta \cdot \sigma_n^2(t_{i-1}) + (1 - \beta) \cdot z^2(t_i) \tag{10}$$

where $\beta = 0.95$ and $d_m(t_i)$ is measured according to (3). This is a typical methodology employed when information about the measurement noise is not available [49].

Finally, regarding the system initial conditions, quick convergence is obtained when the initial system error variance $\boldsymbol{P}(0)$ is much larger than the state noise variance $\boldsymbol{Q}$. We have set:

$$\boldsymbol{P}(0) = \begin{bmatrix} 100 & 0 \\ 0 & 10^{-1} \end{bmatrix}$$

In these conditions, the initial estimate of the state can be freely set to any value.

[7]Consider that rerouting on the Internet is an unlikely event and, as such, bottleneck capacity typically keeps constant throughout the duration of a connection.

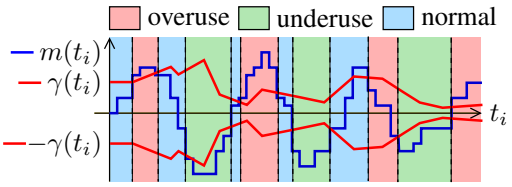


Figure 3. Over-use detector signaling

Congestion detection

In this Section, we design a congestion detection mechanism based on the one-way delay variation $m(t_i)$ estimated by the Kalman filter. To this purpose, we propose to detect congestion by comparing $m(t_i)$ with a threshold $\gamma(t_i)$. Such a mechanism is implemented by the block *over-use detector* shown in Figure 1. In particular, this block produces a signal s which can take three different values (see Figure 3):

1) *overuse* when $m(t_i) > \gamma(t_i)$: an increasing queue length is detected;
2) *underuse* when $m(t_i) < -\gamma(t_i)$: a decreasing bottleneck queue is detected, i.e. the input rate is below the available bandwidth;
3) *normal* when $-\gamma(t_i) \leq m(t_i) \leq \gamma(t_i)$: an unchanged congestion state is detected.

The value of the threshold γ is critical to tune the congestion detection mechanism. Intuitively, a small threshold would make the algorithm very sensitive in detecting changes in the congestion state, but it would have the drawback of being too sensitive to noise. Conversely, a large threshold would result in a sluggish detection of congestion state changes but would be more robust with respect to noise. Moreover, a constant value for the threshold γ cannot be used (see also [50]) because two issues can occur: (i) the delay-based controller may never affect sending rate computation if the size of bottleneck queue is not sufficiently large and (ii) GCC flows are starved in the presence of concurrent loss-based TCP traffic.

To overcome such issues we propose an *Adaptive Threshold* to have a detection mechanism that adapts to network conditions. We propose the following control law to dynamically adapt the threshold:

$$\gamma(t_i) = \gamma(t_{i-1}) + \Delta T_i \cdot k_\gamma(t_i)(|m(t_i)| - \gamma(t_{i-1})) \tag{11}$$

where $\Delta T_i = t_i - t_{i-1}$, and t_i is the time instant the i-th video frame is received. The gain $k_\gamma(t_i)$ is defined as follows:

$$k_\gamma(t_i) = \begin{cases} k_d & |m(t_i)| < \gamma(t_{i-1}) \\ k_u & \text{otherwise} \end{cases} \tag{12}$$

where k_u (k_d) determine the rate at which the threshold is increased (decreased). The threshold $\gamma(t_i)$ is a low-pass

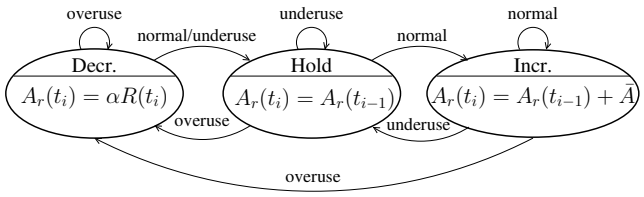


Figure 4. Remote rate controller finite state machine

filtered version of $|m(t_i)|$. The tuning of k_u (k_d) will be discussed in Section IV.

Rate control

A finite state machine (FSM) is driven by the signal s produced by the over-use detector and computes the rate A_r as follows:

$$A_r(t_i) = \begin{cases} A_r(t_{i-1}) + \bar{A} & \text{Increase,} \\ \alpha R_r(t_i) & \text{Decrease,} \\ A_r(t_{i-1}) & \text{Hold,} \end{cases} \quad (13)$$

where t_i denotes the time the i-th *video frame* is received, $\alpha = 0.85$, and $R_r(t_i)$ is the measured received rate. $\bar{A}$ is the increase rate and is set equal to the ratio between half of the average packet size and the round trip time. The finite state machine (FSM) is shown in Figure 4: the state of the FSM (13) is driven by the *over-use detector* signal s which is based on $m(t_i)$. It is important to notice that $A_r(t_i)$ is upper bounded by $1.5R_r(t_i)$.

The rationale of the proposed FSM is the following: when the bottleneck buffer starts to build-up, the estimated one-way delay variation $m(t_i)$ becomes positive. Then, the over-use detector detects this variation and triggers an *overuse* signal, which drives the machine into the "Decrease" state. As a result, the sending rate is reduced and the bottleneck buffer starts to be drained up to the point when the estimated one-way delay variation $m(t_i)$ becomes negative. An *underuse* signal is then triggered, which drives the machine into the "Hold" state. The machine remains in the "Hold" state until the bottleneck buffer gets empty. When this occurs, $m(t_i)$ approaches zero and the overuse detector generates a *normal* signal, which drives the machine into the "Increase" state again.

B. The loss-based controller

The loss-based controller complements the delay-based controller in the case losses are measured. The algorithm acts every time t_k a feedback message carrying A_r is received by the sender. The feedback messages are sent through the real-time control protocol (RTCP). The RTCP reports include the fraction of lost packets $f_l(t_k)$ computed as described in the RTP RFC [51]. The sender uses $f_l(t_k)$ to compute the sending rate $A_s(t_k)$ according to the following equation:

$$A_s(t_k) = \begin{cases} A_s(t_{k-1})(1 - 0.5 f_l(t_k)) & f_l(t_k) > 0.1 \\ 1.05(A_s(t_{k-1})) & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases} \quad (14)$$

The rationale of (14) is simple: (i) when the fraction of lost packets is small ($0.02 \leq f_l(t_k) \leq 0.1$), A_s is kept constant,

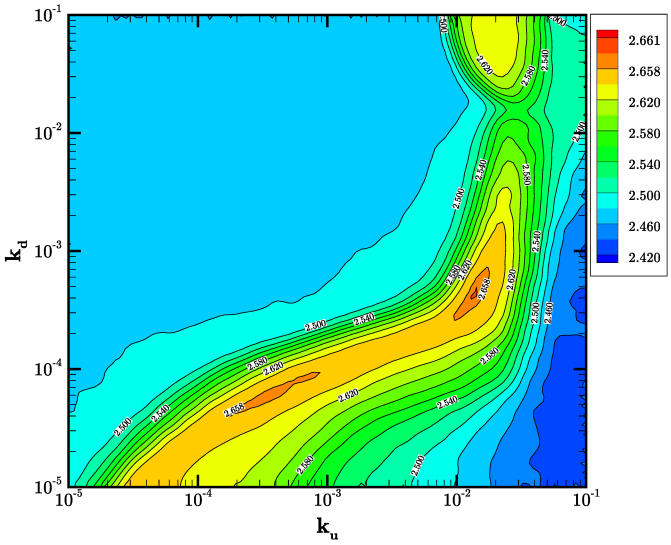


Figure 5. Contour plot of the objective function U as a function of k_u and k_d

(ii) when the fraction of lost packets is high ($f_l(t_k) > 0.1$) the rate is multiplicatively decreased, whereas (iii) when the fraction of lost packets is negligible ($f_l(t_k) < 0.02$), the rate is multiplicatively increased.

IV. The Adaptive Threshold Design

In this Section, we describe the tuning of the adaptive threshold parameters used to detect congestion. In particular, we first motivate the choice of the parameters k_u and k_d used in (12) which determine the speed at which the threshold is increased or decreased. Then, we explain why an adaptive threshold has to be employed to avoid (i) that the delay-based controller may never affect the sending rate computation if the size of the bottleneck queue is not sufficiently large and (ii) the starvation of GCC flows in the presence of concurrent loss-based TCP traffic.

Choice of the threshold parameters. The adaptive threshold dynamics depends on two parameters k_u and k_d which define how quickly the threshold $\gamma(t_i)$ follows the delay variation $m(t_i)$, i.e. $1/k_u$ and $1/k_d$ are the dynamics time constant. In order to tune these parameters we state an optimization problem by employing the objective function proposed in [52]:

$$U(\boldsymbol{x}) = \sum_{i=1}^{N} U_a(x_i) \quad (15)$$

where $U_a(x_i)$ is the objective function measured for the i-th flow. The overall utilization is obtained as the sum of $U_a(x_i)$ for each of the N concurrent flows. x_i is the average throughput of the i-th flow and $U_a(x)$ is a concave utility function given by:

$$U_a(x_i) = \begin{cases} \log(x_i) & a = 1 \\ \frac{x_i^{1-a}}{1-a} & \text{otherwise} \end{cases} \quad (16)$$

It is well-known that for any value of $a > 0$ this optimization problem is Pareto-efficient [53], [52]. The maximization of U implies a fair allocation of the throughput among concurrent flows. In the case of video conferencing we need to extend (15) to also consider the impact of the queuing delay on system performance [30]:

$$U(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{N} (U_a(x_i) - \delta \cdot U_b(y_i)) \tag{17}$$

The same rationale used for $U_a(x_i)$ applies to $U_b(y_i)$ where y_i is the average queuing delay measured for the i-th flow. The parameters a and b express the trade-off between fairness and efficiency for throughput and delay whereas δ expresses the relative importance of delay with respect to throughput. Following [30], we use $a = 2$, $b = 1$, which gives more emphasis on throughput fairness, and $\delta = 0.15$, which guarantees $U_a(x_i)$ and $U_b(y_i)$ are well balanced.

We have considered three scenarios in which we have measured the objective function (17): (i) a single GCC flow over a bottleneck with variable link capacity, similarly to the use case shown in Figure 10, (ii) multiple concurrent GCC flows over a bottleneck with constant link capacity, similarly to the use case depicted in Figure 13, (3) one GCC flow against one TCP flow over a bottleneck with constant link capacity. The objective function has been computed for every value of k_u and k_d in the range $[10^{-5}, 0.1] \times [10^{-5}, 0.1]$ divided into 200×200 intervals in the logarithmic scale. The contour map of the sum of the normalized objective functions U obtained in each scenario for every couple (k_u,k_d) is shown in Figure 5. U increases when $k_u > k_d$, i.e. when the threshold $\gamma(t_i)$ is quickly increased and slowly decreased. However when $k_u \gg k_d$ (in the bottom-right corner of Figure 5) the threshold decreases too slowly reducing the algorithm sensitivity to the delay inflation; this induces higher queuing delays and, as a consequence, U decreases. On the other hand, when $k_u < k_d$ the algorithm becomes very sensitive to the delay variation which leads to throughput degradation in the presence of concurrent TCP flows. The maximum is obtained for (k_u,k_d) $\simeq (0.01, 0.00018)$. This setting provides the best trade-off between throughput, latency, intra and inter-protocol fairness.

Influence of the bottleneck queue size on the delay variation. We now explain why an adaptive threshold is needed. In fact, a constant threshold $\gamma(t_i) = \overline{\gamma}$, may inhibit the delay-based controller if $\overline{\gamma}$ is larger than the maximum measurable delay variation.

Proposition 1: Let us consider one GCC flow accessing a drop-tail bottleneck queue with maximum queuing time $\overline{T}_q$. Over-use signals cannot be generated, and thus the delay-based controller is inactive, if the following condition holds:

$$\overline{\gamma} > \sqrt{2\frac{\overline{T}_q}{\tau}} + \frac{\overline{T}_q}{\tau}, \tag{18}$$

where τ is the time constant of the exponential increase phase of the sending rate computed using the loss-based algorithm (14).

Proof: We start by recalling that the goal of the delay-based controller is to stop the increase of the sending rate (14) before the queuing delay gets too large. Towards this end, the overuse detector of the delay-based controller compares the estimated $m(t)$ with a static threshold $\overline{\gamma}$ and it triggers a decrease of the rate if $m(t) > \overline{\gamma}$. Thus, if the maximum value m_M of $m(t)$ is less than $\overline{\gamma}$ the generation of the over-use signal is inhibited. Therefore, to prove this proposition we need to show that if (18) holds, it turns out that $m_M < \overline{\gamma}$.

We start by computing $m_M = \max(m(t))$. From the hypothesis given in Section III, $m(t)$ is an estimate of the queuing delay gradient $\dot{T}_q(t)$. Since we are interested in finding the maximum of $m(t)$, we only consider the exponential increase phase of the sending rate $r(t)$ that holds when the measured loss rate is less than $f_l(t) < 0.02$ according to (14). A fluid-flow model of the increase phase of the sending rate $r(t)$ can be easily derived from (14) as follows:

$$\dot{r}(t) = \frac{1}{\tau} r(t) \tag{19}$$

Now, by combining (2) and (1) we obtain:

$$m(t) = \frac{\dot{q}(t)}{C} = \begin{cases} \frac{r(t)}{C} - 1 & 0 \le q(t) \le q_M \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Let $t_0 = 0$ denote the time instant when $r(t_0) = C$, i.e. the instant after which the queue starts to build up, and let us analyze the dynamics of $m(t)$ for $t > t_0$. Under these assumptions, by integrating (19) it turns out

$$r(t) = r(t_0) \exp(t/\tau) = C \cdot \exp(t/\tau) \tag{21}$$

Now, by substituting (21) in $m(t) = r(t)/C - 1$, we obtain:

$$m(t) = \exp(t/\tau) - 1. \tag{22}$$

Eq. (22) is a monotonically increasing function until the point $q(t) = q_M$, i.e. when the queue is full and packets start to get dropped. Thus, the maximum value of $m(t)$ is equal to $m(t_M)$ where t_M is the time instant at which the queue becomes full, i.e. $q(t_M) = q_M$. By considering (2) and integrating between t_0 and t_M we obtain:

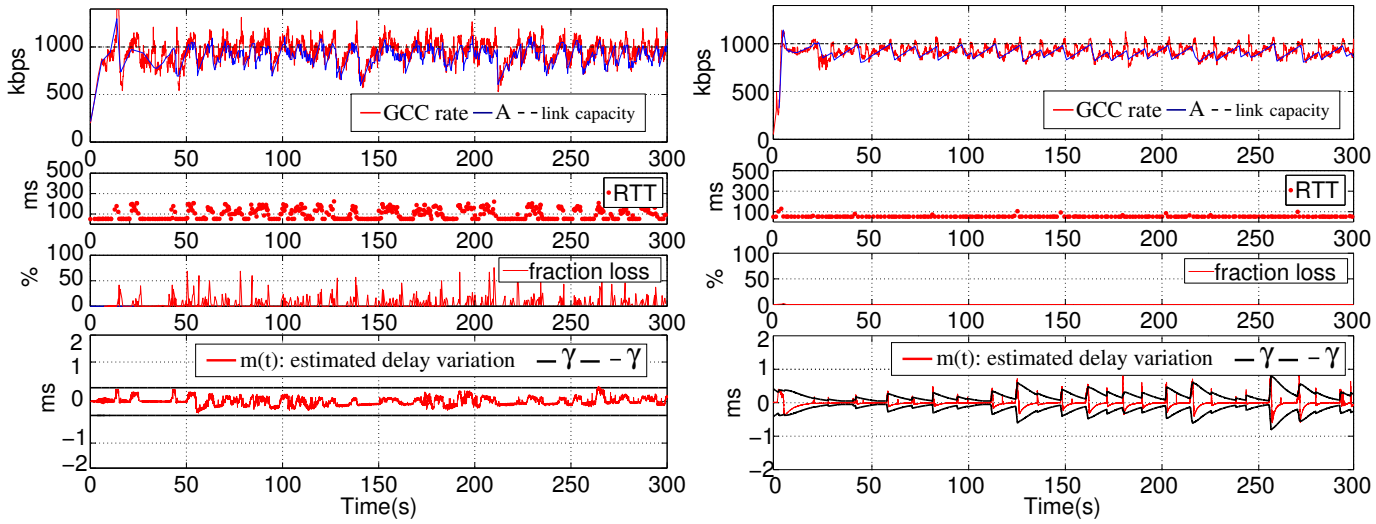
$$q(t_M) = q_M = \int_{t_0}^{t_M} (r(\xi) - C) d\xi = C\tau(\exp(t_M/\tau) - 1) - Ct_M. \tag{23}$$

Based on the experimental evaluation of Section VI we measured that τ is in the order of seconds and is much larger than t_M; thus by computing the second order McLaurin expansion for the exponential and substituting it in (23) we obtain the following approximation:

$$t_M = \sqrt{\frac{2q_M}{\tau C}}.$$

Thus, the maximum queuing delay is:

$$m_M = m(t_M) = \frac{1}{\tau}\left(\sqrt{\frac{2\tau q_M}{C}} + \frac{q_M}{C}\right) = \sqrt{2\frac{\overline{T}_q}{\tau}} + \frac{\overline{T}_q}{\tau}. \tag{24}$$



(a) Static threshold

(b) Adaptive threshold

Figure 6. Effect of the adaptive threshold in the case of a single GCC flow over a 1 Mpbs link with queue size $\overline{T}_q = 150$ ms

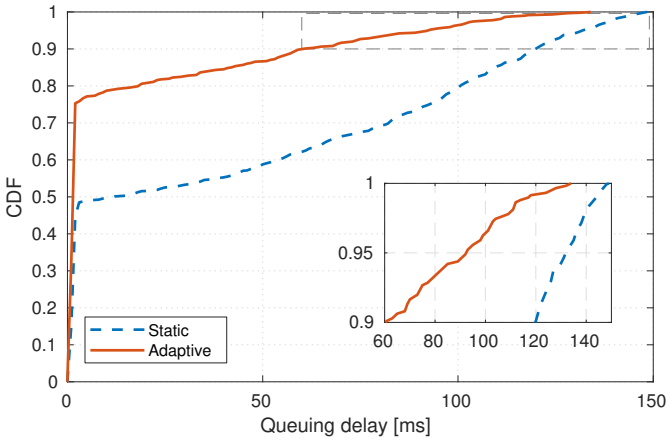


Figure 7. RTT comparison in the case of a single GCC flow

The proposition is proved by observing that if the condition (18) holds, it turns out that $m_M < \overline{\gamma}$. ∎

To experimentally validate the issue of using a constant threshold, Figure 6 (a) shows a real experiment (see Section V for details on the testbed) where a single GCC flow is started over a 1 Mbps bottleneck link with a drop-tail queue whose maximum queuing time is $\overline{T}_q = 150$ ms. Figure 6 compares the GCC rate, the RTT, and the fraction of lost packets in the case of a static setting of the threshold (Figure 6 (a)) and in the case of the adaptive threshold (Figure 6 (b)). With a static threshold $\overline{\gamma}$, the delay-based controller becomes ineffective and is not able to react in the presence of delay inflation. This is due to the fact that the maximum value of the delay variation, which depends on $\overline{T}_q$, is smaller than $\overline{\gamma}$. On the other hand, Figure 6 (b) shows that using the adaptive threshold, $\gamma(t)$ follows $m(t)$ with a slower time constant and, when $m(t)$ overshoots $\gamma(t)$, the delay-based algorithm can reduce the sending rate. Figure 6 (b) also shows that the controller is able to avoid packet losses, i.e. $f_l(t) = 0$ throughout the whole duration of the experiment. To further show the benefits of the adaptive threshold, Figure 7 compares the cumulative distribution function of the measured RTT in the two experiments reported in Figure 6. The measured median value of the RTT is very close to the propagation delay $RTT_{\min} = 50$ ms, whereas thanks to the adaptive threshold 5th percentile of the RTT is reduced from 130 ms to 90 ms.

Effect of a concurrent TCP flow on the delay variation. In this paragraph, we show that the threshold must be adaptive to avoid the starvation of a GCC flow in the presence of a concurrent loss-based flow. Towards this end, we consider a single GCC flow with a concurrent long-lived TCP flow. We show that a static setting of the threshold might lead to the starvation of the GCC flow. In this scenario, the one-way delay variation can be expressed as the sum of two components:

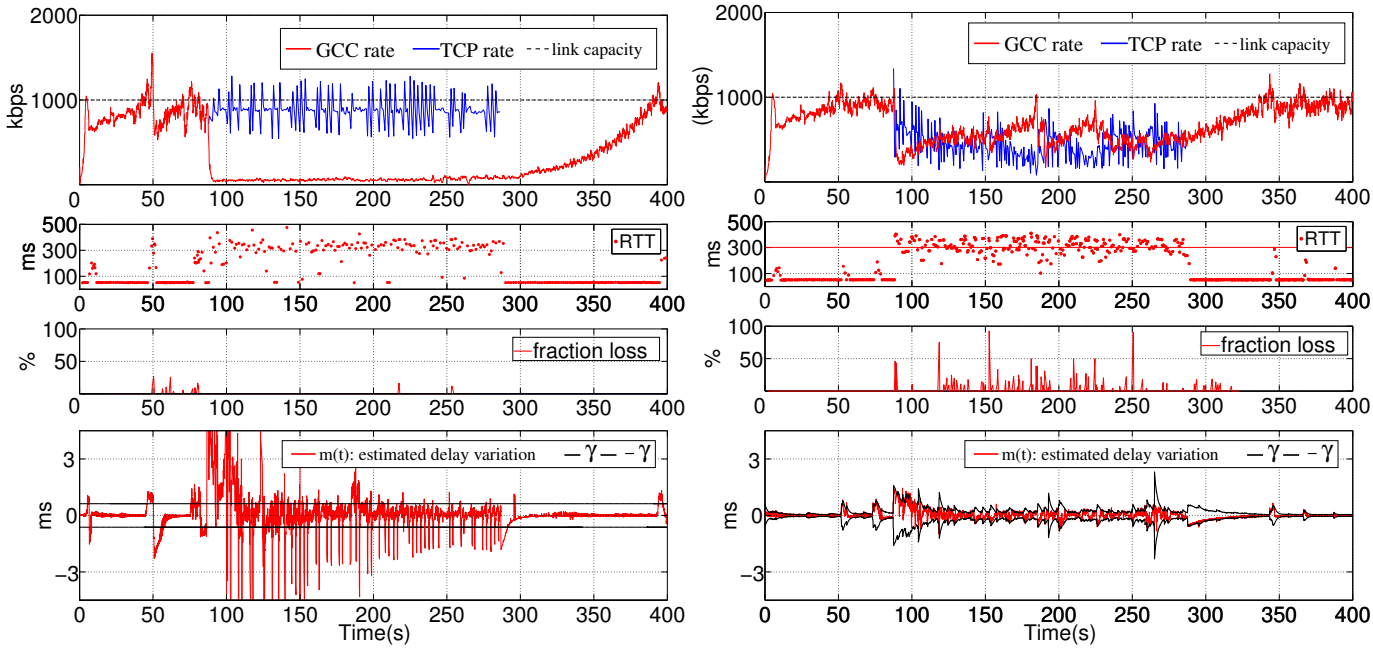
$$m(t) = m_1(t) + m_2(t) = \frac{r_1(t) + r_2(t)}{C} - 1 \qquad (25)$$

where $m_1(t)$ and $m_2(t)$ are the queuing delay variations of the GCC and the TCP flow respectively and $r_1(t)$ and $r_2(t)$ are the corresponding sending rates.

In the following, we show that the maximum delay variation $m_{2,M}$ due to a TCP flow can be much larger than that of a GCC flow. In particular, by using similar arguments employed to derive (24) we obtain:

$$m_{2,M} = \frac{\max(\dot{q}(t))}{C} = \frac{\max(r_2(t)) - C}{C}. \qquad (26)$$

A well-known and widely used approximation of the TCP throughput is $r_2(t) = w(t)/RTT(t)$ [44], where $w(t)$ is the congestion window of the TCP flow and $RTT(t)$ is the round trip time. The maximum value that the congestion window can assume is the queue size q_M plus the in-flight bytes $C \cdot RTT_{\min}$ [45], i.e. $\max(w(t)) = q_M + C \cdot RTT_{\min}$, whereas the minimum value of $RTT(t)$ is the round trip



(a) Static threshold

(b) Adaptive threshold

Figure 8. One GCC flow vs one TCP flow. Bottleneck parameters: queue size $\overline{T}_q = 350$ms, link capacity $C = 1000$kbps

propagation $RTT_{\min}$. Thus, it turns out that $\max(r_2(t)) = q_M/RTT_{\min} + C$ which yields to:

$$m_{2,M} = \frac{q_M}{RTT_{\min} \cdot C} = \frac{\overline{T}_q}{RTT_{\min}}, \tag{27}$$

The ratio between $m_{2,M}$ and $m_{1,M}$ is given by:

$$\frac{m_{2,M}}{m_{1,M}} = \frac{1}{RTT_{\min}} \frac{\overline{T}_q}{\sqrt{2\frac{\overline{T}_q}{\tau} + \frac{\overline{T}_q}{\tau}}} \tag{28}$$

It is clear that, when $\overline{T}_q$ increases, the ratio monotonically increases. Since in this case $m(t)$ contains the component $m_2(t)$ due to the TCP flow, if $m_2(t) \gg m_1(t)$ the GCC flow will decrease A_r not because of self-inflicted delay, but due to the queuing delay provoked by the TCP flow. This means that using a static threshold $\overline{\gamma}$, the TCP flow would starve the GCC flow when large queues are used.

To experimentally validate the theoretical findings, Figure 8 shows a real experiment which has been run by employing the testbed described in Section V. Figure 8 shows a GCC flow which competes with a TCP flow over a 1 Mbps bottleneck link with a drop-tail queue whose maximum queuing time is $\overline{T}_q = 350$ ms. Figure 8 (a) shows that, when a static threshold is used, the GCC flow gets starved. In particular, when the TCP flow is started, $m(t)$ begins to oscillate above the threshold $\overline{\gamma}$ mainly due to the queuing delay variation induced by the TCP flow $m_2(t)$, which triggers a large number of overuse signals. Consequently, the remote rate controller FSM enters the decrease mode which reduces the value of A_r according to (13). On the other hand, Figure 8 (b) shows that the adaptive threshold avoids the starvation and provides fairness between GCC and TCP flow. In particular, after TCP is started, $\gamma(t)$ follows $m(t)$ with a smaller time constant which avoids the generation of numerous consecutive overuse signals and prevents the starvation of the GCC flow.

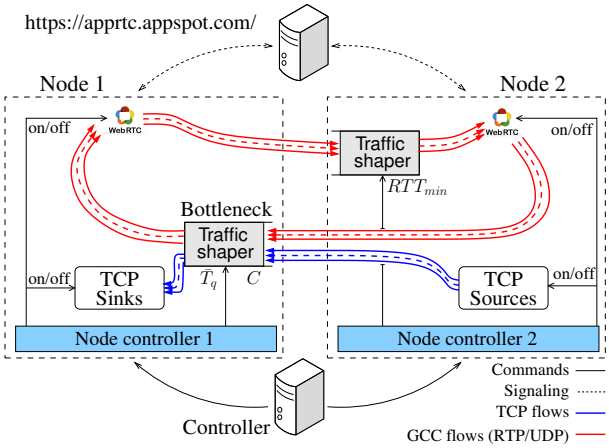


Figure 9. Experimental testbed

V. Testbed

Figure 9 shows the experimental testbed employed to emulate a WAN scenario. Further details on how to reproduce the experiments are available on-line [15]. The testbed consists of four Linux machines equipped with a Linux kernel 3.16.0. Two nodes, each one running several sessions of Chromium browsers [14] and an application to generate or receive TCP long-lived flows, are connected through an Ethernet cable. Thus, the behavior of GCC flows when competing against TCP flows can be analyzed. Another node runs a web server which handles the signaling required to establish the video calls between the browsers. The last node is the testbed *controller*

that orchestrates the experiments via `ssh` commands. The testbed controller undertakes the following tasks: (i) it places the WebRTC calls starting the GCC flows; (ii) it sets the link capacity C and the bottleneck queue size $\overline{T}_q$ on Node 1; (iii) it sets the propagation delay $RTT_{\min}$ on Node 2; (iv) it starts the TCP flows when required.

The bottleneck queue is placed on Node 1 and employs a drop-tail queuing discipline[8]. The queue size q_M has been set by considering the maximum time required to drain the buffer $\overline{T}_q$, i.e. $q_M = \overline{T}_q \cdot C$. The round trip propagation delay $RTT_{\min} = 50$ ms, which is the sum of the propagation delays on the direct and reverse path (25 ms each) has been set on Node 2 through the NetEm Linux module, whereas the bottleneck queue size $\overline{T}_q$ has been set in the range $[150,\ 700]$ ms and C in the range $[500,\ 6000]$ kbps according to a large measurement study on the edge network [54]. We have used a Token Bucket Filter (TBF) to set the ingress link capacity C of Node 1.

Video and TCP settings. The TCP sources employ the CUBIC congestion control which is the default in Linux kernel. The congestion window, the slow-start threshold, the RTT, and the sequence number are logged. A Web server[9] provides the HTML page that handles the signaling between the peers using the WebRTC JavaScript API [11]. The same video sequence is used to enforce experiments reproducibility. To this purpose, we have used the *"Four People"*[10] YUV test sequence which is cyclically repeated. Chromium encodes the raw video source with the VP8 video encoder[11]. We have measured that, without any bandwidth limitation, VP8 limits the sending bitrate $A_s(t)$ to a maximum value of 2Mbps.

Metrics. In order to quantitatively assess the performance of GCC we consider QoS metrics such as packet loss ratio, average bitrate, and delay which are known to be well correlated with QoE metrics through, for instance, the IQX hypothesis [55]. Following this approach has the advantage of using metrics that are not sensitive to application specific aspects, such as the employed video encoder. Moreover, splitting the evaluation of QoE metrics from QoS metrics also follows the guidelines defined within the IETF RTP Media Congestion Avoidance Techniques (RMCAT) working group [13]. In particular, we consider:

- **Channel Utilization** $U = R_r/C$, where C is the known link capacity and R_r is the measured average received rate;
- **Loss ratio** $l = (\text{bytes lost})/(\text{bytes sent})$;
- **5th, 25th, 50th, 75th and 95th percentile of queuing delay**, measured as $RTT(t) - RTT_{\min}$ over all the RTT samples reported in the RTCP feedback during the experiments;
- **Jain's Fairness Index**: $J_{\text{FI}}(t) = \frac{(\sum_{i=1}^{N} x_i(t))^2}{N \sum_{i=1}^{N} x_i(t)^2}$, where $x_i(t)$ is the measured instantaneous throughput of the i-th flow and N is the total number of competing flows.

[8]The interplay of GCC with other queuing disciplines has been separately addressed in [42].

[9]https://apprtc.appspot.com/

[10]https://people.xiph.org/~thdavies/x264_streams/FourPeople_1280x720_30/

[11]http://www.webmproject.org/

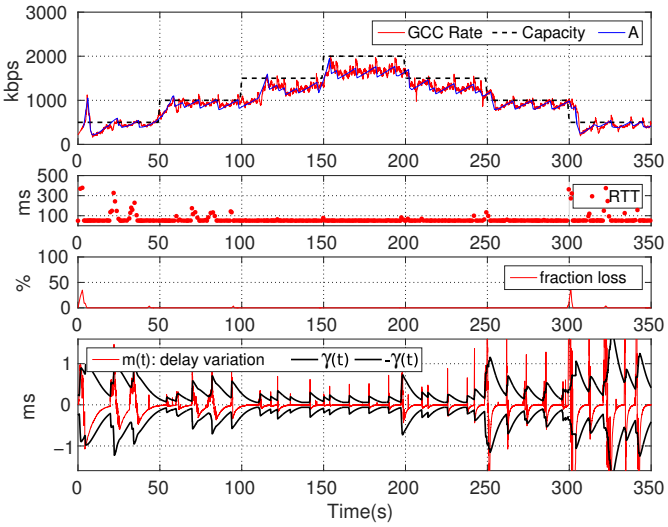


Figure 10. GCC Rate, fraction loss, RTT and delay variation dynamics in the case of a single GCC with variable link capacity

VI. Experimental Evaluation

In this Section, we present the experimental results obtained by employing the testbed described in Section V. The goal is to check if GCC satisfies the real-time communication requirements defined in [43] i.e., low queuing in the absence of concurrent heterogeneous traffic and a reasonable share of bandwidth when competing with other homogeneous or heterogeneous flows. In particular, we have considered a scenario where a single GCC runs under different bottleneck conditions, a scenario where GCC shares the bottleneck with long and short-lived TCP flows, and finally a scenario where a variable number of GCC flows share the bottleneck. It is important to notice that in a typical video conferencing session peers are connected to edge networks and bottlenecks are generally in the uplink interfaces [54].

A. Single GCC flow

We start the experimental evaluation by considering a single GCC flow over a bottleneck. We consider two cases: the first one aiming at checking how GCC adapts its sending rate when step-like changes of the link capacity occur; the second one refers to a home network scenario in which only a video conference session is using the network.

Variable link capacity. In this Section, we investigate how a GCC flow adapts its sending rate when step-like changes of the link capacity occur. In particular, the link capacity C varies as a staircase: starting from C equal to 500 kbps, C is increased every 50 s of 500 kbps until a capacity of 2000 kbps is reached. Then, C is decreased using the same pattern. The round trip propagation delay $RTT_{\min}$ has been set to 50 ms. Figure 10 shows that the GCC sending rate is able to quickly match the variable link capacity C. Furthermore, GCC contains the queuing delay since the RTT is kept close

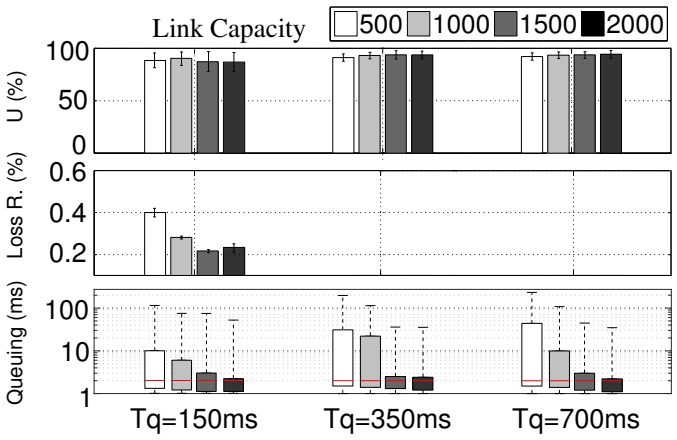


Figure 11. Channel utilization, Loss ratio and Queuing delay percentiles in the case of a single GCC flow over a bottleneck with different constant capacity $C \in \{500, 1000, 1500, 2000\}$ kbps and bottleneck queue size $\overline{T}_q \in \{150, 350, 700\}$ ms

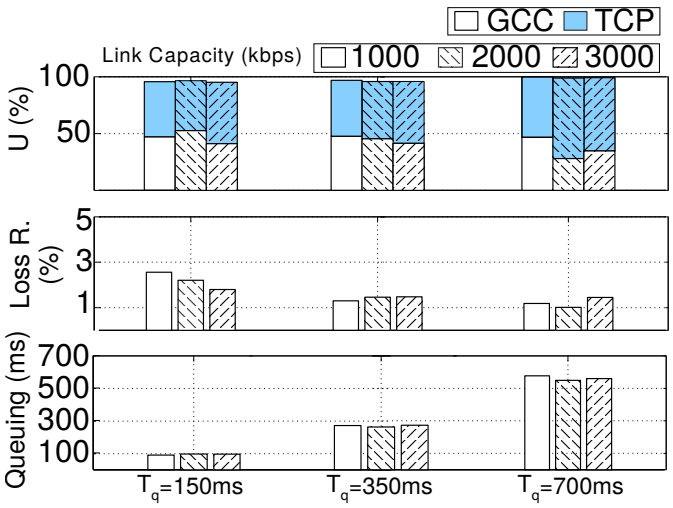


Figure 12. Channel utilization, Loss ratio and Queuing delay in the case of a GCC flow with a TCP flow over a bottleneck with different constant capacity $C \in \{1000, 2000, 3000\}$ kbps and bottleneck queue size $T_q \in \{150, 350, 700\}$ ms

to $RTT_{\min}$ during the entire video call. Overall, the average measured channel utilization is roughly equal to 86%. We can conclude assessing that GCC is able to track the link capacity while limiting packet loss and maintaining low queuing delays.

Different bottleneck conditions with constant link capacity. In this Section we investigate the performance of a single GCC flow over a bottleneck with a constant link capacity. The capacity C of the bottleneck has been set to $C \in \{500, 1000, 1500, 2000\}$ kbps and three values of the queue size have been considered, i.e. $\overline{T}_q \in \{150, 350, 700\}$ ms. For each of the considered 12 couples (C, $\overline{T}_q$), we have run 10 video calls whose duration is 300 s and we have evaluated the metrics defined in Section V by averaging over the 10 experiments. Figure 11 shows the metrics grouped by the value of the queue size $\overline{T}_q$; each group contains a bar which shows the metric for a particular value of C in terms of the average channel utilization U, the average loss ratio and queuing delay percentiles. The channel utilization is slightly above 90% in every experiment regardless the parameters C and $\overline{T}_q$. GCC is able to avoid losses when $\overline{T}_q = 350$ ms or $\overline{T}_q = 700$ ms whereas in the case of $\overline{T}_q = 150$ ms some losses are measured. Queuing delays are depicted using a box and whisker plot in logarithmic scale: the bottom and top of the box are respectively the 25th and 75th percentile, whereas the red band in the box is the median; the end of the whiskers represents the 5th and 95th percentile. Let us focus on the influence of the link capacity C on the queuing delays. We notice that in all the experiments 95th percentile is larger as the link capacity decreases: this is due to the fact that the arrival filter measures a larger delay variation at lower link capacity according to equation (1). Let us focus on the influence of the bottleneck queue size $\overline{T}_q$. When $\overline{T}_q = 150$ ms, the 95th percentile for $C = 500$ kbps is limited by the small queue size at the expense of some losses due to overflow. As the value of $\overline{T}_q$ increases, its influence on the queuing delays becomes less remarkable, proving that GCC is able to properly contain queuing delays. Moreover, Figure 11 shows that the median value of queuing is maintained below 3 ms in every experiment.

B. GCC flow with TCP flows

We now consider the case of one GCC flow when sharing the bottleneck link with one TCP flow. This scenario aims at verifying if GCC is able to fairly share the link capacity with a long-lived TCP flow under different bottleneck conditions.

In particular, this case considers the scenario where during a video conference session a user shares a large file with the other peer. The video call starts at $t = 0$ s and lasts 400 s whereas the TCP flow is active for 200 s in the time interval $[100, 300]$ s as shown in Figure 8. The capacity has been set to $C \in \{1000, 2000, 3000\}$ kbps and the bottleneck queue size to $\overline{T}_q \in \{150, 350, 700\}$ ms [54]. For each of the 9 couples (C, $\overline{T}_q$), we have run 10 experiments measuring the metrics defined in Section V by averaging over the results obtained in the 10 experiments. Figure 12 shows the metrics grouped by the value of the queue size $\overline{T}_q$; each group contains a bar which shows the metric for a particular value of C in terms of the average channel utilization U, the average loss ratio and the average value of queuing delay. In this case, we use the average value of the queuing delay instead of the percentiles since the TCP flow tends to fill the bottleneck queue; as expected this results in measuring an average queue occupancy roughly equal to its maximum length $\overline{T}_q$. Figure 12 shows that the link capacity is fairly shared among GCC and TCP flows when $\overline{T}_q = 150$ ms or $\overline{T}_q = 350$ ms whereas TCP slightly prevails over GCC when the bottleneck queue size increases to $\overline{T}_q = 700$ ms. This is made possible by the adaptive threshold that, by reducing the sensitivity of the delay-based controller, leads the GCC flow to be controlled by both the loss-based and delay-based algorithms in the presence of concurrent loss-based traffic. In fact, the value of the fraction loss measured for the GCC flow for any value of the couples (C, $\overline{T}_q$) is greater than zero confirming that the algorithm is also operating in loss-based mode. One experiment dynamics

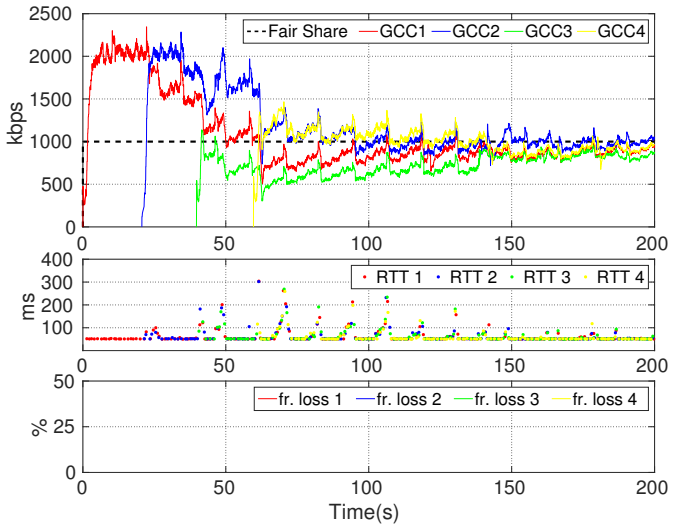


Figure 13. GCC Rate, fraction loss, and RTT dynamics in the case of four concurrent GCC flows over a 4 Mbps link

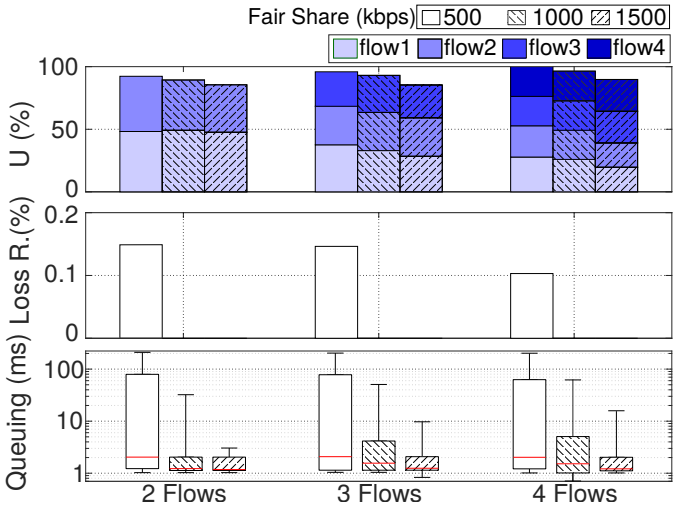


Figure 14. Channel utilization, Loss ratio and Queuing delay percentiles in the case of variable number of concurrent GCC flows $n \in \{2, 3, 4\}$ over a bottleneck with different fair share $F_s \in \{500, 1000, 1500\}$ kbps

has been previously shown in Figure 8(b) to describe the adaptive threshold in Section IV.

To conclude this section, we point out that the same set of experiments has also been run by employing the NewReno congestion control in the place of CUBIC. The results obtained with NewReno are comparable to those presented above for CUBIC and, due to the lack of space, are not reported in this paper. Nevertheless, a complete performance evaluation, in the future, should also include different TCP congestion control algorithms and consider flows with different round-trip times to check for RTT fairness.

C. Multiple concurrent GCC flows

The aim of this Section is to investigate the GCC intra-protocol fairness. This case considers the scenario where video conference is used by more than one user, for example in a home network scenario. Toward this end, we consider a variable number of concurrent GCC flows $n \in \{2, 3, 4\}$ and the link capacity C varies in such a way that the fair share F_s of the capacity among the flows is equal to $F_s = C/n \in \{500, 1000, 1500\}$ kbps, f.i. if $n = 2$, C is set to $C = F_s \cdot n \in \{1000, 2000, 3000\}$; the bottleneck queue size has been set to $\overline{T}_q = 350$ ms in every experiment. Each flow is started 20 s after the previous one and the experiment lasts 200 s. For each of the 9 couples (n, F_s) we have run 10 experiments and evaluated the metrics by averaging over the 10 experiments. Figure 13 shows the dynamics of one experiment where four GCC flows share the bottleneck with a fair share F_s set to 1 Mbps (i.e. C =4 Mbps). This experiment shows that GCC is not affected by the *"late-comer effect"* [25]; the three GCC flows fairly share the link and the measured Jain Fairness Index approaches 0.93. No packet is lost during the whole duration of the experiment. Figure 14 summarizes the results obtained for each couple (n, F_s). The results are grouped by the value of the number of concurrent GCC flows n; each group contains a bar which shows the metric for a particular value of F_s in terms of the average channel utilization U, the average loss ratio and queuing delay percentiles in logarithmic scale. First of all, we notice that the number of concurrent flows does not remarkably influence the metrics: the JFI measured in every experiment is above 0.9, confirming that the algorithm provides intra-protocol fairness, and the cumulative channel utilization is always kept above 85%. Some losses are measured only when $F_s = 500$ kbps. For what concerns the queuing delays the results are similar to the ones obtained in Section VI-A in the case of a single GCC flow: the 95th percentile is larger as the link capacity decreases whereas the median value is always kept below 3 ms exactly as in the case of the single flow scenario. Overall we can conclude that GCC is able to provide intra-protocol fairness while maintaining at the same time low queuing delay and losses.

VII. Conclusions

In this paper we have presented and evaluated the Google Congestion Control (GCC) algorithm which is used to control the sending rate of video conferencing applications. GCC has been implemented in the open-source Chromium Web browser and it has been adopted by the Google Chrome browser. An extensive experimental evaluation has shown that GCC is able to contain queuing delays while providing intra and inter protocol fairness along with full link utilization.

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology 2013-2018," *White Paper*, Jun. 2014.

[2] "Global Internet Phenomena Report, 2014," https://www.sandvine.com/trends/global-internet-phenomena/.

[3] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM CCR*, vol. 36, no. 1, pp. 59–62, Jan. 2006. [Online]. Available: http://doi.acm.org/10.1145/1111322.1111336

[4] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: Reducing the flow completion time tail in datacenter networks," in *Proc.of ACM SIGCOMM*, Helsinki, Finland, Aug. 2012, pp. 139–150. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342390

[5] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2149–2196, thirdquarter 2016.

[6] E. Brosh, S. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The Delay-Friendliness of TCP for Real-Time Traffic," *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1478–1491, Oct. 2010.

[7] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *ACM Transaction Multimedia Computer Communication Applications*, vol. 4, no. 2, pp. 16:1–16:22, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1352012.1352020

[8] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (dccp)," Internet Requests for Comments, RFC Editor, RFC 4340, March 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4340.txt

[9] L. De Cicco and S. Mascolo, "A mathematical model of the Skype VoIP congestion control algorithm," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 790–795, Mar. 2010.

[10] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 826–839, Jun. 2014.

[11] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "Webrtc 1.0: Real-time communication between browsers," *W3C Working Draft*, Feb. 2015.

[12] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and Design of the Google Congestion Control for Web Real-time Communication (WebRTC)," in *Proc. of the ACM Multimedia Systems Conference*, Klagenfurt, Austria, May 2016. [Online]. Available: http://dx.doi.org/10.1145/2910017.2910605

[13] Z. Sarker, V. Singh, X. Zhu, and M. Ramalho, "Test Cases for Evaluating RMCAT Proposals," Work in progress, (Work in progress), Internet-Draft draft-ietf-rmcat-eval-test-04, Oct. 2016.

[14] "Chromium Git repositories." [Online]. Available: https://chromium.googlesource.com/

[15] "Google Congestion Control Testbed settings." [Online]. Available: http://c3lab.poliba.it/index.php?title=WebRTC_Testbed

[16] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM SIGCOMM CCR*, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[17] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. of Workshop on Protocols for Fast Long-Distance Networks*, vol. 4, Feb. 2004.

[18] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM CCR*, vol. 34, no. 2, pp. 85–96, Apr. 2004.

[19] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM CCR*, vol. 34, no. 2, pp. 25–38, Apr. 2004.

[20] S. Mascolo and F. Vacirca, "The effect of reverse traffic on the performance of new TCP congestion control algorithms," in *proc. of International Workshop on Protocols for Fast Long-distance Networks*, Nara, Japan, 2006.

[21] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[22] K. Jacobsson, L. L. H. Andrew, A. Tang, S. H. Low, and H. Hjalmarsson, "An Improved Link Model for Window Flow Control and Its Application to FAST TCP," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 551–564, Mar. 2009.

[23] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (ledbat)," Internet Requests for Comments, RFC Editor, RFC 6817, December 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6817.txt

[24] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internets with heterogeneous transmission media," in *Proc. of International Conference on Network Protocols*, Oct. 1999, pp. 213–221.

[25] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The Quest for LEDBAT Fairness," in *IEEE Global Telecommunications Conference*, Dec. 2010, pp. 1–6.

[26] D. A. Hayes and G. Armitage, "Revisiting TCP Congestion Control Using Delay Gradients," in *Proc. of the 10th IFIP TC 6 Conference on Networking - Volume Part II*, Jul. 2011, pp. 328–341. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008826.2008858

[27] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," in *Proc. of ACM SIGCOMM*, vol. 45, no. 5, Aug. 2015, pp. 509–522. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787498

[28] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *Proc. of the ACM SIGCOMM*, 2015, pp. 537–550. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787510

[29] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks," in *Proc. of USENIX NSDI*, Apr. 2013.

[30] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-generated Congestion Control," in *Proc. of ACM SIGCOMM*, Aug. 2013.

[31] X. Zhu, R. Pan, S. Mena, P. Jones, J. Fu, and S. D'Aronco, "NADA: A unified congestion control scheme for real-time media," Work in progress, (Work in progress), Internet-Draft draft-ietf-rmcat-nada-04, Mar. 2017.

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

[33] S. Holmer, H. Lundin, G. Carlucci, L. De Cicco, and S. Mascolo, "Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web," Work in progress, (Work in progress), Internet-Draft draft-ietf-rmcat-gcc-02, Jul. 2016.

[34] L. De Cicco, G. Carlucci, and S. Mascolo, "Congestion control for webrtc: Standardization status and open issues," *IEEE Communications Standards Magazine*, Jun. 2017.

[35] D. Wischik and N. McKeown, "Part i: Buffer sizes for core routers," *ACM SIGCOMM CCR*, vol. 35, no. 3, pp. 75–78, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1070873.1070884

[36] G. Raina, D. Towsley, and D. Wischik, "Part ii: Control theory for buffer sizing," *ACM SIGCOMM CCR*, vol. 35, no. 3, pp. 79–82, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1070873.1070885

[37] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Part iii: Routers with very small buffers," *ACM SIGCOMM CCR*, vol. 35, no. 3, pp. 83–90, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1070873.1070886

[38] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," Internet Requests for Comments, RFC Editor, RFC 3168, September 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3168.txt

[39] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue, ACM*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2208917.2209336

[40] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Comm. of the ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012.

[41] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "Pie: A lightweight control scheme to address the bufferbloat problem," in *Proc. of IEEE HPSR '13*, Jul. 2013.

[42] G. Carlucci, L. De Cicco, and S. Mascolo, "Controlling Queuing Delays for Real-Time Communication: The Interplay of E2E and AQM Algorithms," *ACM SIGCOMM CCR*, Jul. 2016.

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

[44] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 945–959, Jun. 2002.

[45] S. Mascolo, "Congestion control in high-speed communication networks using the Smith principle," *Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.

[46] O. Gurewitz, I. Cidon, and M. Sidi, "One-way delay estimation using network-wide measurements," *IEEE/ACM Transactions on Networking*, vol. 14, no. SI, pp. 2710–2724, Jun. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.2006.874414

[47] S. Biaz and N. H. Vaidya, "Is the round-trip time correlated with the number of packets in flight?" in *Proc. of ACM IMC*, 2003, pp. 273–278. [Online]. Available: http://doi.acm.org/10.1145/948205.948240

[48] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[49] R. Bos, X. Bombois, and P. M. Van den Hof, "Designing a Kalman filter when no noise covariance information is available," in *Proc. of IFAC World Congress*, vol. 16, Jul. 2005, pp. 212–212.

[50] L. De Cicco, G. Carlucci, and S. Mascolo, "Understanding the Dynamic Behaviour of the Google Congestion Control for RTCWeb," in *Proc. of Packet Video Workshop*, San Jose, CA, dec 2013.

[51] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," Internet Requests for Comments, RFC Editor, STD 64, July 2003. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3550.txt
[52] R. Srikant, *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.
[53] F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low, "Congestion control for high performance, stability, and fairness in general networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 43–56, Feb 2005.
[54] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: illuminating the edge network," in *Proc. of ACM IMC*, 2010, pp. 246–259.
[55] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A generic quantitative relationship between quality of experience and quality of service," *IEEE Network*, vol. 24, no. 2, pp. 36–41, 2010.

Gaetano Carlucci received the Telecommunications Engineering degree (Hons.) from Technical University of Bari, Bari, Italy, in 2011 and the Ph.D. degree in Information and Communication Technologies Engineering in March 2015 from "Scuola Interpolitecnica Di Dottorato" a joint PhD program of high qualification among the three Italian Polytechnic Universities. Currently, he is a Post-Doc at Politecnico di Bari. He has held a visiting position at the Video and Content Platforms Research and Advanced Development Group at Cisco Systems 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.

Luca De Cicco (M' 14) received the computer science engineering degree (Hons.) and the Ph.D. degree in information engineering from the Polytechnic of Bari, Bari, Italy, in 2003 and 2008, respectively. Currently, he is an Assistant Professor at Politecnico di Bari. He has held visiting positions at the University of New Mexico, Albuquerque, NM, USA, in 2007; Ecole Superieure d'Electricité, Paris, France, in 2012; and the Laboratory of Information, Networking and Communication Sciences-LINCS, Paris, France, in 2013 and 2014. He is the co-author of more than 40 papers published in international journals, books, or conferences. His main interests are the modeling and design of congestion control algorithms for multimedia transport, adaptive video streaming, Software Defined Networks, and Server Overload Control. Currently, he is an Associate Editor of the Internet Technology Letters journal (Wiley)

Stefan Holmer received his M. Sc. degree in Computer Science and Electrical Engineering from Linköping University in 2008. He has been working as a research and software engineer at Google since 2011, specializing in networking, bandwidth estimation and video processing for real-time applications. Prior to Google he worked on real-time communication at Global IP Solutions.

Saverio Mascolo (SM' 07) received the Laurea degree (Hons.) in Electronics Engineering and the Ph.D. both from Politecnico di Bari, Italy, in 1991 and 1994, respectively. Currently he is Full Professor and Chair of the Department of Electrical Engineering and Computer Science at Politecnico di Bari. He was a Postdoctoral Researcher in 1995 and a Visiting Researcher in 1999 at the University of California, Los Angeles (UCLA) and Visiting Consultant at the University of Uppsala, Sweden, from 2002 to 2004. He has authored or co-authored more than 120 papers in international journals, books, or conferences. He holds four U.S. and three Italian patents. He has worked on intelligent manufacturing systems, deadlock avoidance, nonlinear control, chaotic systems, synchronization of chaotic systems using observers, crypto communications using observers, modelling and control of data networks, congestion control, adaptive video streaming, content delivery networks, software-defined networks and server overload control. He has been Associate Editor of the IEEE Transactions on Automatic Control. Currently, he is Associate Editor of IEEE/ACM Transactions on Networking and of Computer Networks Journal, Elsevier.