Controlling Queuing Delays for Real-Time Communication: The Interplay of E2E and AQM Algorithms

Gaetano Carlucci
Politecnico di Bari, Italy
gaetano.carlucci@poliba.it

Luca De Cicco
Politecnico di Bari, Italy
luca.decicco@poliba.it

Saverio Mascolo
Politecnico di Bari, Italy
saverio.mascolo@poliba.it

ABSTRACT

Real-time media communication requires not only congestion control, but also minimization of queuing delays to provide interactivity. In this work we consider the case of real-time communication between web browsers (WebRTC) and we focus on the interplay of an end-to-end delay-based congestion control algorithm, i.e. the Google congestion control (GCC), with two delay-based AQM algorithms, namely CoDel and PIE, and two flow queuing schedulers, i.e. SFQ and Fq_Codel. Experimental investigations show that, when only GCC flows are considered, the end-to-end algorithm is able to contain queuing delays without AQMs. Moreover the interplay of GCC flows with PIE or CoDel leads to higher packet losses with respect to the case of a DropTail queue. In the presence of concurrent TCP traffic, PIE and CoDel reduce the queuing delays with respect to DropTail at the cost of increased packet losses. In this scenario flow queuing schedulers offer a better solution.

CCS Concepts

•Networks → Network protocol design; •Information systems → Web conferencing;

Keywords

WebRTC, Congestion Control, Active Queue Management

1. INTRODUCTION

Containing delays is important for a wide spectrum of Internet applications ranging from data center transport to Real Time Communication (video conferencing) and live video streaming. Since adding bandwidth can be considered relatively cheap, Internet latency is today the main obstacle to improving the performance of these applications [3]. Delays are due to propagation time, that can contribute with up to roughly 150 ms in planetary networks, and to queuing time, that may lead to delays in the order of seconds when excessively large buffers are employed in access networks (*bufferbloat*) [8, 14].

In the context of delay-sensitive applications, video conference has particularly challenging requirements in terms of interactivity and bandwidth. In fact, such applications require minimization of delays and packet losses in order to respectively enhance interactivity and avoid media quality degradation. These requirements can be met by employing two complementary control algorithms: one placed at the end points, namely the *end-to-end* congestion control algorithm; the other placed in the network routers, i.e. the *Active Queue Management* (AQM) algorithm.

When designing end-to-end congestion control algorithms for video conference, UDP is a natural choice, since such applications cannot tolerate large latencies due to TCP packet retransmissions. Additionally, delay-based congestion control algorithms are usually preferred to loss-based ones since they can detect congestion before packets are lost due to buffer overflow.

On the other hand, AQM schemes control the router buffers by dropping the packets or marking them if ECN is used [21]. Despite the fact that many AQM algorithms have been proposed over the past two decades [7], the adoption of these algorithms has been hampered by two main issues [19]: 1) they aim at controlling the average queue length which is not well correlated with network congestion and 2) an ad-hoc configuration of their parameters has to be made. These issues, along with the *bufferbloat* phenomenon, have motivated the study of new AQM algorithms, such as CoDel [19] and PIE [20], that do not require parameter tuning and that explicitly control the queuing delay instead of the queue length.

Several papers have studied the interactions between 1) TCP algorithms and AQMs [12, 15, 10], 2) data center transports and network prioritization [18], 3) low priority congestion control and AQMs [9].

In this paper, we consider the interplay of end-to-end delay-based congestion control for real-time communication and AQM algorithms placed in routers. This research topic is particularly timely in view of 1) the increased deployment of new AQM algorithms in operational networks (see f.i. [10]) and 2) the emerging WebRTC framework which is an enabling technology to allow real-time communication among users through Web browsers. To the purpose, we have considered PIE and CoDel as representative of parameterless AQM algorithms and the Google Congestion Control (GCC) since it is the only widely deployed algorithm which adheres to the WebRTC framework. In particular, GCC is implemented in the Google Chrome browser which is today beyond any doubt the most used browser[1].

We analyze this interaction in two key scenarios: 1) a video flow in isolation, and 2) a video flow with a varying number of TCP flows. Results show that, in the first scenario, the interplay between GCC and AQMs is not always beneficial since AQMs induce losses on the video flow not experienced under DropTail with a queue size accommodating 300 ms worth of packets (DT/300) [22].

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

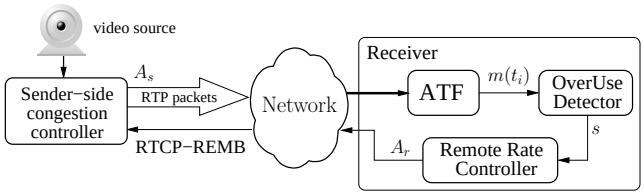


Figure 1: Google congestion control architecture

When concurrent TCP traffic is considered, both PIE and CoDel are able to effectively reduce the queuing delays compared to DT/300, but they provoke packet losses on the video flow that increase with the number of concurrent TCP flows. This issue is alleviated by flow queuing schedulers which provide isolation among the flows avoiding complex dynamic interaction.

2. ALGORITHMS

This Section briefly describes the algorithms involved in our study. Section 2.1 reviews CoDel and PIE along with Fq_CoDel and SFQ. Section 2.2 describes the Google Congestion Control (GCC) which is used in Google Chrome browsers to implement congestion control functionalities in the WebRTC stack and in Google Hangouts.

2.1 AQM Algorithms

2.1.1 CoDel

The goal of CoDel (*Controlled Delay Active Queue Management*) [19] is to contain the queuing latency while maximizing the throughput. CoDel does not require any parameters tuning and it has been designed to work across a wide range of conditions with different links and round trip times. CoDel works as follows: a timestamp is added to each packet at the ingress side of the queue in order to measure the packet *sojourn time* at the egress side when packets are dequeued. The *sojourn time* is then compared to a *target delay* (5 ms by default)*;* if it is below the *target* the packet is forwarded, otherwise the algorithm starts a timer and forwards the packet. When packets are dequeued, CoDel checks the *sojourn time*, and if it gets below the *target*, the timer is stopped. However, if the timer reaches the value of *interval* (100 ms by default), CoDel enters the dropping state which is left when the sojourn time gets below the *target delay*. During this state, whenever the *interval* timer expires a packet is dropped, the timer is reset, and the next timer duration is set to $interval \leftarrow interval/\sqrt{N+1}$, where N is the cumulative number of packets dropped while in dropping state. Thus, the longer the sojourn time stays above the target delay, the higher the packet dropping frequency. Finally, as soon as the measured *sojourn time* of an outgoing packet gets below the *target*, the dropping state is left and *interval* is restored to its default value.

2.1.2 PIE

Similarly to CoDel, PIE (*Proportional Integral controller Enhanced*) [20] aims at controlling the queuing latency and does not require parameters tuning. PIE employs a proportional-integral (PI) controller that computes a random probability to mark or drop packets, based on a target queuing latency. The drop probability p is computed as follows:

$$p = p + \alpha \cdot (del_{est} - target_{del}) + \beta \cdot (del_{est} - old_{del}) \quad (1)$$

where del_{est} is the queuing delay estimated based on the queue occupancy, $target_{del}$ is the target delay (20 ms by default) and old_{del} is the value of the queuing delay estimated in the previous updating. The value of p is updated every T_{update} (30 ms by default). α and β are respectively the integral and proportional gain of the PI controller. PIE is able to dynamically adapt these values based on the level of congestion which is expressed by the value of p: if the network is congested, α and β are increased to make the controller reaction faster. In [20] it is argued that PIE requires less computational efforts compared to CoDel. In fact, PIE does not need to track the per-packet sojourn time since it marks or drops packets on arrival (enqueue time) differently from CoDel that drops packets on departure.

2.1.3 SFQ

SFQ [16] assigns flows in a fixed set of queues serviced in strict round-robin order; the maximum number of queues is configurable (1024 by default in the Linux implementation). In order to assign a queue to an ingress packet, a hash function is applied to its 5-tuple defined by the IP source and destination, layer 4 port source and destination and layer 4 protocol number; packets with the same hash are assigned to the same queue.

2.1.4 Fq_CoDel

Fq_CoDel [11] is a hybrid scheme combining flow scheduling with active queue management. It aims at keeping queue lengths short while providing isolation for low-rate traffic such as DNS, web, and videoconferencing traffic. It consists of a set of queues and a scheduler that decides from which queue a packet should be dequeued. CoDel is employed at each queue. The dequeuing process is byte-based employing a deficit round-robin mechanism.

2.2 Google Congestion Control

Figure 1 shows the architecture of the end-to-end Google Congestion Control (GCC) algorithm [4]. The sender employs a UDP socket to send RTP packets and receive RTCP feedback reports from the receiver. The algorithm has two components: 1) a *delay-based* controller, placed at the receiver, that computes a rate A_r that is fed back to the sender with the aim of keeping the queuing delay small; 2) a *loss-based* controller, placed at the sender, that computes the target sending bitrate A_s that cannot exceed A_r.

The sender-side congestion control. It is a *loss-based* congestion control algorithm that acts every time t_k the k-th RTCP report message arrives at the sender or every time t_r the r-th REMB[2] message, which carries A_r, arrives at the sender. The RTCP reports include, among other feedback information, the fraction of lost packets $f_l(t_k)$ computed as described in the RTP RFC. Based on $f_l(t_k)$, the controller computes the rate $A_s(t_k)$, measured in kbps, according to the following equation:

$$A_s(t_k) = \begin{cases} A_s(t_{k-1})(1 - 0.5 f_l(t_k)) & f_l(t_k) > 0.1 \\ 1.05 A_s(t_{k-1}) & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases} \quad (2)$$

[2]http://tools.ietf.org/html/draft-alvestrand-rmcat-remb-03

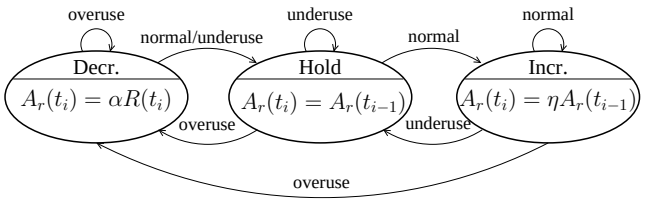


Figure 2: Remote rate controller

The rationale of (2) is simple: 1) when the fraction of lost packets is considered small ($0.02 \leq f_l(t_k) \leq 0.1$), A_s is kept constant, 2) if a high fraction lost is estimated ($f_l(t_k) > 0.1$) the rate is multiplicatively decreased 3) when the fraction lost is considered negligible ($f_l(t_k) < 0.02$), the rate is multiplicatively increased. After A_s is computed through (2), it is set as $A_s \leftarrow \min(A_s, A_r)$ to avoid that A_s exceeds the last received value of A_r.

The receiver-side controller. This controller is made of the three components shown in Figure 1. Each time t_i a group of RTP packets that forms the i-th video frame is received, the *one-way queuing delay variation* $m(t_i)$ is estimated by the *arrival-time filter* (ATF). In order to estimate $m(t_i)$, the ATF starts by measuring the *one way delay variation* $d_m(t_i) = t_i - t_{i-1} - (T_i - T_{i-1})$, where T_i is the time at which the i-th video frame has been sent and t_i is the time at which it has been received. The one way delay variation is considered as the sum of three components [4]: 1) the *transmission time variation*, 2) the *one-way queuing time variation* $m(t_i)$, and 3) the *network jitter* $n(t_i)$. The following mathematical model of the one way delay variation is assumed:

$$d(t_i) = \frac{\Delta L(t_i)}{C(t_i)} + m(t_i) + n(t_i) \quad (3)$$

where $\Delta L(t_i) = L(t_i) - L(t_{i-1})$, $L(t_i)$ is the i-th video frame size, $C(t_i)$ is an estimate of the bottleneck link capacity, and $n(t_i)$ is the network jitter modeled as a Gaussian noise. A Kalman filter is used to extract $m(t_i)$ from the measured one way delay variation.

Then, the *over-use detector* compares the estimated one-way queuing delay variation $m(t_i)$ with an adaptive threshold $\gamma(t_i)$ proposed in [5]: when $m(t_i)$ gets above $\gamma(t_i)$, the network is considered congested and the *overuse* signal is generated; on the other hand, if $m(t_i)$ decreases below $-\gamma(t_i)$, the network is considered underused and the *underuse* signal is generated; when $m(t_i)$ falls back in $[-\gamma(t_i), \gamma(t_i)]$ a *normal* signal is produced.

Finally, the signal s is fed to the *remote rate controller* which drives the finite state machine (FSM) shown in Figure 2 whose goal is to empty the queues along the end-to-end path. A_r is increased (*Increase* state), decreased (*Decrease* state) or kept constant (*Hold* state) depending on its state. In particular, A_r is set according to the equations shown in the states of Figure 2, where $\eta \in [1.005, 1.3]$, $\alpha \in [0.8, 0.95]$, and $R(t_i)$ is the receiving rate measured in the last 500 ms. It is worth noticing that A_r cannot exceed $1.5R(t_i)$. The computed rate A_r is sent to the sender through REMB messages.

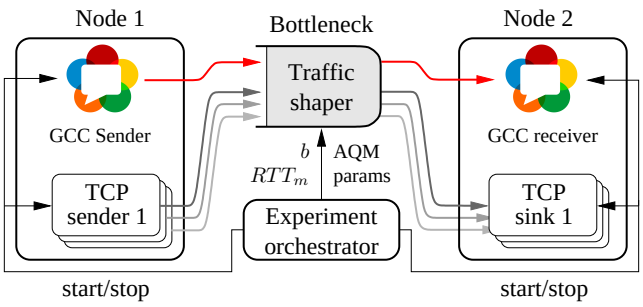


Figure 3: Experimental testbed

3. EXPERIMENTAL TESTBED

In this section, we describe the experimental scenario. More details on how to reproduce the experiments are available on-line[3].

Testbed architecture. Figure 3 shows an essential view of the experimental testbed. Four Linux machines, equipped with a Linux kernel 3.16.0 natively supporting all the AQM algorithms considered in the experiments, are employed. Two machines (Node 1 and Node 2) are connected through an Ethernet cable running two Chromium browsers[4] each and an IPerf-like application to generate or receive a configurable number of TCP long-lived flows; a third machine (not shown in the Figure) runs a web server which handles the signaling required to establish the video calls between the browsers. The fourth machine is the *experiment orchestrator* that performs experiment automation. The propagation delay on node Node 1 has been set to 50 ms through the NetEm Linux module. We have used a Token Bucket Filter (TBF) to set the ingress link capacity b of Node 2. We have turned off the NIC optimization parameters that could interfere with the experiments, i.e. TCP segmentation offload, jumbo frame, generic segmentation offload.

Video and TCP settings. The TCP sources employ the CUBIC congestion control, the default in Linux kernels, and log the congestion window, the slow-start threshold, the RTT, and the sequence number. A Web server[5] provides the HTML page that handles the signaling between the peers using the WebRTC JavaScript API. The same video sequence is used to enforce experiments reproducibility. To the purpose, we have used the Linux kernel module `v4l2loopback`[6] to create a virtual webcam device which cyclically repeats the *"Four People"*[7] YUV test sequence. Chromium encodes the raw video source with the VP8 video encoder[8]. We have measured that, without any bandwidth limitation, VP8 limits the sending bitrate $A_s(t)$ to a maximum value of 2 Mbps.

[3]http://c3lab.poliba.it/index.php?title=WebRTC_Testbed
[4]http://code.google.com/p/chromium/
[5]https://apprtc.appspot.com/
[6]https://github.com/umlaeute/v4l2loopback
[7]https://people.xiph.org/~thdavies/x264_streams/FourPeople_1280x720_30/
[8]http://www.webmproject.org/

Algorithm	Parameter	Value
DropTail (DT/300)	queue size	300 ms
Codel	interval	100 ms
	target	13 ms at 1 Mbps, 5 ms otherwise
	limit	1000 pkts
PIE	t_{update}	30 ms
	t_{target}	20 ms
	limit	1000 pkts
SFQ	queue size	300 ms
Fq_Codel	target	13 ms
	interval	100 ms
	limit	10240 pkts

Table 1: AQM schemes parameter

AQM algorithm settings. We have employed the settings reported in Table 1 as suggested in [12]. In the case of CoDel, the suggested *target* value is 13ms when the link capacity is 1Mbps otherwise the default value of 5 ms is used. Regarding PIE, we have used the default tuning parameters employed in the Linux implementations. Only in Section 4.1 we have varied the *target* parameter of CoDel and PIE to perform a sensitivity analysis. In the case of DropTail and SFQ we have set the queue size to 300 ms, which is the time taken to drain the queue when it is completely full, as suggested in the IETF RMCAT draft [22] for the evaluation of congestion control algorithms for real-time communication.

Metrics. In order to quantitatively assess the interaction between GCC flows and AQM/flow schedulers, 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 [6]. Following 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. Moreover, splitting the evaluation of QoE metrics from QoS metrics also follows the guidelines recently defined within the IETF RTP Media Congestion Avoidance Techniques (RMCAT) working group. In particular, we consider: 1) *Channel Utilization* $U = R/b$, where b is the known link capacity and R is the average received rate; for every experiment we measure the average value and the standard deviation; 2) *Loss ratio* l = (byte lost)/(byte sent); for every experiment we measure the cumulative value; 3) *Queuing delay* $T_q(t_k) = RTT(t_k) - RTT_m$, measured each time t_k an RTCP sample is received, where $RTT(t_k)$ is the k-th RTT sample and RTT_m is the known round trip propagation delay; for every experiment we compute the average value, the standard deviation, the 5th, 25th, 50th, 75th and 95th percentile.

4. EXPERIMENTS

In this section, we investigate how the performance of real-time video flows is impacted by the interaction of GCC congestion control algorithm employed at the end points, and several AQMs employed at the bottleneck queue. Throughout all this section we consider the DropTail (DT/300) queuing discipline as the baseline for performance comparison.

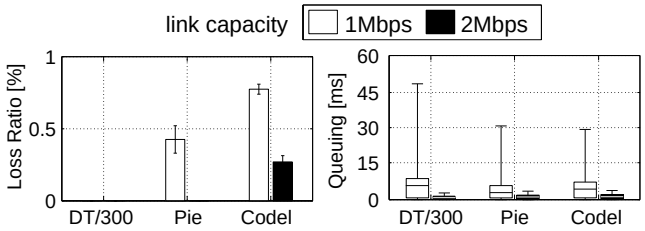


Figure 4: Loss ratio (average and standard deviation) and queuing delay (box plot) in the case of a single GCC flow

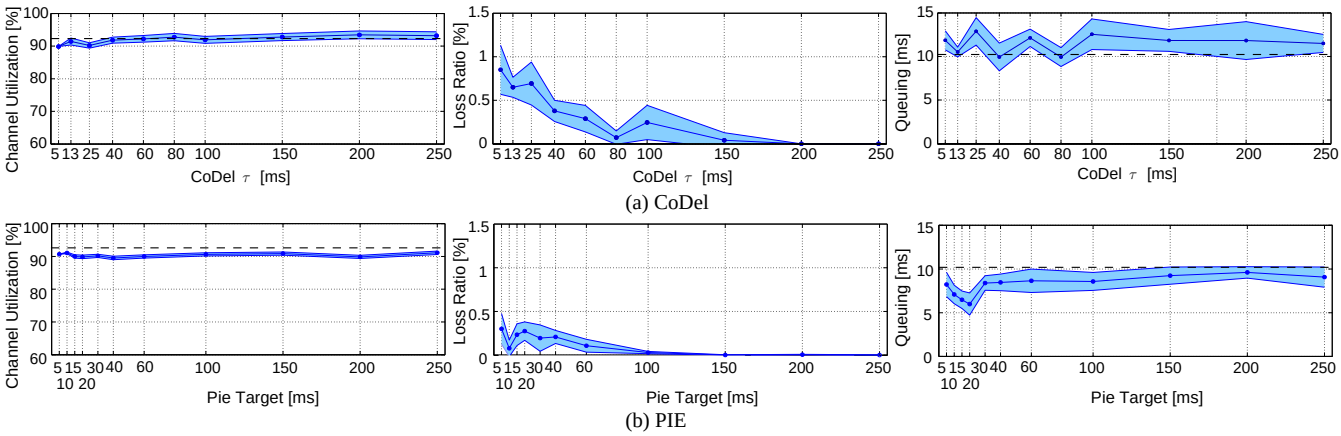
The analysis is divided into two parts. The first part considers the case of a single real-time video flow accessing a bottleneck governed by either DT/300, CoDel or PIE. The second part considers the case of one GCC flow sharing the bottleneck with a varying number of TCP long-lived flows. This is a particularly challenging scenario for real-time flows since it is well-known that when delay-based flows share a bottleneck with loss-based ones, they are typically penalized in terms of channel utilization and delays. Thus, in order to provide a complete picture, in addition to DT/300, PIE, CoDel, we also consider the SFQ scheduling algorithm and Fq_CoDel to quantify the benefits brought by flow isolation.

Finally, we remark that also the case of multiple concurrent GCC flows has been considered. Since these results do not add any new insights, they have been omitted due to space constraint.

4.1 Video flow in isolation

We start the analysis by comparing the video metrics obtained when one video flow runs in isolation over a bottleneck governed by either DT/300, PIE, or CoDel using the settings reported in Table 1. We have considered two values for the link capacity $b \in \{1, 2\}$ Mbps and the round trip propagation delay has been set to 50 ms. We have purposely considered the case of a link capacity $b = 2$ Mbps matching the maximum video encoder bitrate (see Section 3). For each combination of this setting, we have run 6 experiments establishing video calls with a duration of 300 seconds.

The results of the experiments are shown in Figure 4 and grouped in accordance with the queue management discipline. Regarding the packet loss ratio, average values, and standard deviation are shown. Queuing delays are depicted using a box and whisker plot: the bottom and top of the box are respectively the 25-th and 75-th percentile, whereas the band in the box is the median; the end of the whiskers represent the 5-th and 95-th percentile. Even though Figure 4 does not depict it due to lack of space, a channel utilization higher than 90% has been measured across all combinations of bottleneck capacity and queuing discipline. Figure 4 shows that median queuing delays obtained under both PIE and CoDel are only negligibly lower than the one obtained under DT/300. The only improvement provided by CoDel and PIE is a reduction of the 95-th percentile from 48ms to around 30ms. However, if decreasing the tail latency is crucial in many interactive services such as data center transport, gaming, and web search, video conferencing is more tolerant to tail latency and less to losses [3]. Median queuing delays obtained with



(a) CoDel

(b) PIE

Figure 5: Sensitivity analysis: impact of CoDel and PIE targets on GCC flow metrics (average and standard deviation)

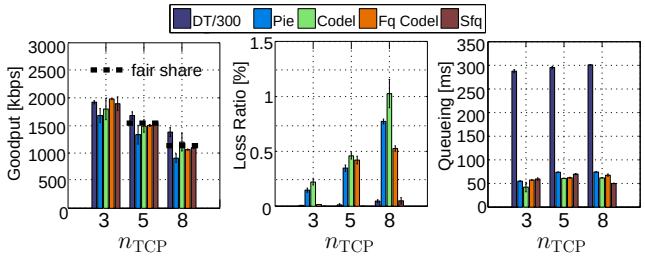


Figure 6: A single video flow with n_{TCP} concurrent TCP flows over a 10 Mpbs bottleneck link

PIE and CoDel are similar to those of DT/300 due to the fact that GCC is able to effectively contain the queuing thanks to its delay-based congestion controller. Let us now consider packet losses: in the case of DT/300 no losses are measured regardless of the bottleneck capacity; PIE provokes slightly less than 0.5% packet losses in the case of a 1Mbps link and, as expected, does not provoke any losses in the case of $b = 2$Mbps; CoDel provokes consistently the highest losses, roughly 0.75% in the case of $b = 1$ Mbps and less than 0.5% in the case of $b = 2$ Mbps. This shows that CoDel is too aggressive and provokes losses even when link capacity is 2 Mbps, i.e. when it is equal to the maximum sending rate that can be produced by the application.

The findings described above motivate us to dig deeper and perform a sensitivity analysis in which AQM algorithm parameters are varied. With this purpose we let the CoDel target τ_{CoDel}, whose default value is 5 ms, vary in the set:

$$\mathscr{T}_{\mathrm{CoDel}} = \{\mathbf{5}, 13, 25, 40, 60, 80, 100, 150, 200, 250\}\,\mathrm{ms}$$

whereas we let the PIE target τ_{PIE}, whose default value is 20 ms, vary in the set:

$$\mathscr{T}_{\mathrm{PIE}} = \{5, 10, 15, \mathbf{20}, 30, 40, 60, 100, 150, 200, 250\}\,\mathrm{ms}$$

For each value of the targets 6 runs of a duration of 300 s have been carried out in the case of a constant link capacity of 1 Mbps. The results are shown in Figure 5 that depicts average and standard deviation of the metrics defined in Section 3 as a function of AQMs targets. The dashed lines shown in the figure represent the corresponding value of the metric measured in the baseline case of DT/300. Figure 5 (a) shows the results obtained when CoDel is employed: even though channel utilization is always kept above 90%, the highest values are the ones measured when τ_{CoDel} is high, i.e. when CoDel is less aggressive in controlling the delay. In fact, when τ_{CoDel} is close to its default value of 5 ms, the interaction of GCC and CoDel becomes detrimental in terms of utilization and loss ratio. This is due to the fact that CoDel reacts to the delay inflation before GCC and induces losses on the video flow. Consequently, GCC works in the loss-based mode which leads to a slightly lower sending bitrate. Finally, queuing delays are always kept on average below 15 ms regardless of the value of τ_{CoDel}.

Figure 5 (b) shows the sensitivity with respect to PIE target τ_{PIE}. Overall, packet losses decrease from less than 0.5% for $\tau_{\mathrm{PIE}} = 5$ ms to zero, recovering the DT/300 performance, when the target is larger than 150 ms (roughly one order of magnitude larger than the default). The queuing is only slightly decreased when τ_{PIE} is small, i.e. when PIE drop packets in order to keep the queuing delay under the set-point τ_{PIE}. Interestingly, the minimum of queuing delay is obtained in correspondence of the default value of 20 ms used in the Linux implementation.

To conclude, this scenario shows that the queuing delay can be successfully contained by employing the GCC delay-based congestion control. In this case, AQMs degrade performance since they introduce packet losses.

4.2 Video flow versus multiple TCP flows

This section considers the case of one real-time video flow when competing with a number n_{TCP} of concurrent TCP flows. We consider AQM algorithms, namely PIE and CoDel, or packet schedulers, i.e. SFQ and Fq_CoDel. We employ the same settings of Section 4.1 and shown in Table 1.

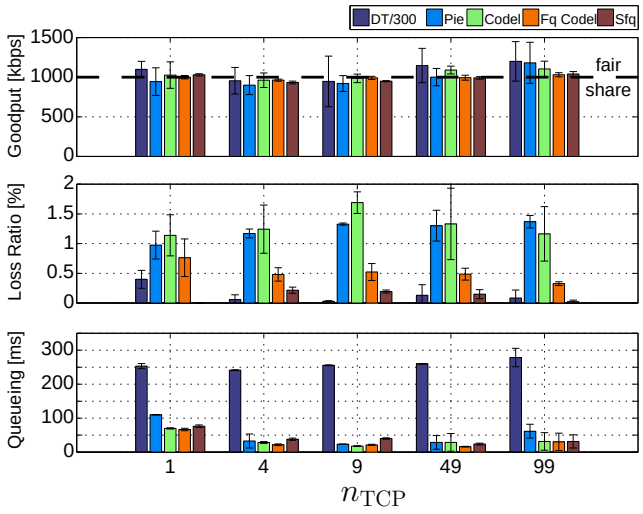


Figure 7: A single GCC flow with n_{TCP} concurrent TCP flows

We have taken into account two cases: 1) $n_{\mathrm{TCP}} \in \{3, 5, 8\}$ with link capacity kept constant at 10Mbps; 2) $n_{\mathrm{TCP}} \in \{1, 4, 9, 49, 99\}$ with the link capacity $b \in \{2, 5, 10, 50, 100\}$ Mbps set in such a way that the fair share $b/(n_{\mathrm{TCP}} + 1)$ for each experiment is constant and equal to 1 Mbps. For each combination of these settings 6 runs have been carried out by establishing video calls with a duration of 400 seconds and started at $t = 0$. TCP flows are started at $t = 100$ s and stopped at $t = 300$ s. Results are shown in Figure 6 and Figure 7 where each bar represents the average value and the error bar is the standard deviation; bars are grouped in accordance with the number of concurrent TCP flows. Figure 6 and Figure 7 show that the goodput obtained by the video flow is always kept around the fair share, even in the case of DT/300 thanks to the GCC adaptive threshold design [5]. Let us now focus on the loss ratio. Under DT/300, the loss ratio is contained in the range $]0, 0.5]$% for any number of concurrent TCP flows; this means that GCC, in order to fairly share the bandwidth with the TCP flow, has been driven in the loss-based mode in which GCC behaves more aggressively in order to fairly share the link with TCP. Video packet losses are higher than 1% when PIE or CoDel are used. In particular, Figure 6 shows that, when increasing the number of concurrent TCP flows, induced losses under PIE, CoDel and Fq_CoDel are increased which indeed is detrimental for real-time video quality.

Let us now consider the queuing delays. In the case of DT/300 a queuing delay of 300 ms is obtained. This is expected since TCP fills the pipe and the queue size has been set in such a way that the maximum queuing delay is 300 ms. All the other considered algorithms were able to contain the queuing delay around 50 ms in the case of $n_{TCP} = 1$ or less than 50 ms with more concurrent TCP flows. Overall, SFQ provided the best results since it achieved the same queuing delay of PIE, CoDel, and Fq_Codel, but with remarkably lower packet losses.

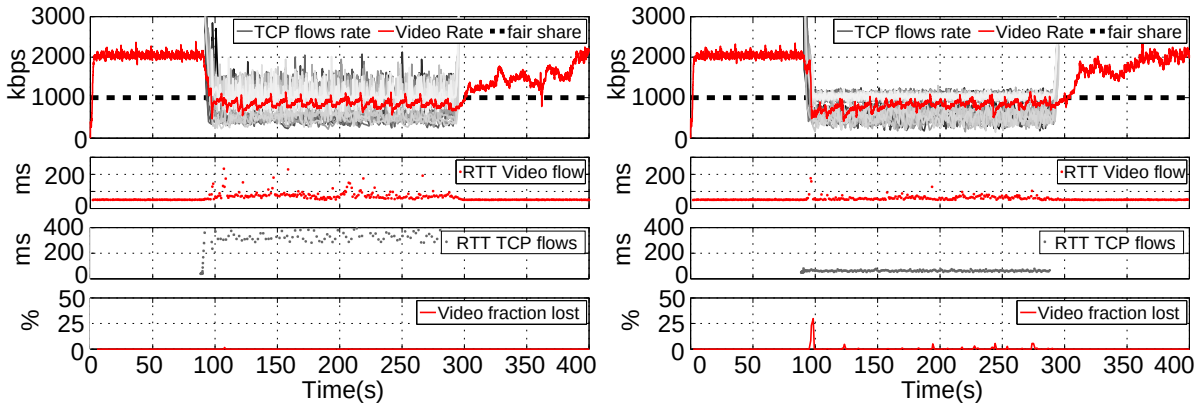
To get a further insight on the different behavior of GCC with SFQ or with Fq_Codel in the presence of concurrent TCP traffic, Figure 8 compares the dynamics of rates, RTT and fraction loss in the two cases when $n_{\mathrm{TCP}} = 99$ and $b = 100$ Mbps. As expected, in both cases the video flow reaches the fair share at 1 Mpbs with an RTT small enough to guarantee real-time interaction. However, the GCC flow does not experience any losses in the case of SFQ, whereas Fq_CoDel provokes losses due to the CoDel algorithm.

5. RELATED WORK

Most of the literature on the evaluation of AQM algorithms considers standard TCP flows in order to measure metrics such as TCP throughput or induced queuing delays [19, 20, 15]. In [12] an extensive experimental evaluation has been carried out sending real traffic, comprising Web pages, VoIP and TCP bulk transfer over residential links governed by a wide selection of bottleneck queue management. The evaluation shows that AQMs help to reduce queue latency induced by TCP, whereas flow queuing schedulers alleviate performance issues due to concurrent flows interaction providing the best balance in terms of throughput and queuing latency. In [10] authors study the performance of modern AQM schemes deployed over a DOCSIS cable modem environment through Ns-2 simulations. A rich traffic mix is considered, including FTP, adaptive video streaming over HTTP, and VoIP. The study finds that AQMs remarkably improve applications performance compared to Drop Tail queues. A different conclusion is drawn in [2] where the author points out that application performance may degrade due to the losses induced by PIE and CoDel since they focus only on keeping the queuing delay under a constant target.

Only a few publications considered the interactions of AQMs and delay-based congestion control algorithms. In [9] an undesired behavior is described when queue management algorithms interact with bitTorent flows which employ a delay-based congestion control algorithm named LEDBAT [23]. LEDBAT is meant to contain the queuing delay and yield bandwidth in presence of concurrent TCP flows. Authors have found that, when queue management algorithms are present on the path, an undesired phenomenon occurs causing LEDBAT flows to fairly share the bandwidth with TCP flows. The closest work to ours is [13] that considers SVC-based Congestion Control (SCC), a recently proposed delay-based congestion control algorithm for interactive video, and TCP CUBIC in the case of queues governed either by DropTail or PIE. Ns-2 simulation results show that SCC is not able to grab the fair share when competing with TCP flows in the case DropTail queues are employed.

In this work we consider GCC, a congestion control algorithm that today is employed in Google Chrome and Google Hangouts to transport video conferencing traffic. At the best of our knowledge, this is the first work that analyzes the interplay of AQMs and real video conference traffic. Finally, we point out that this work does not consider the potential benefit of employing explicit congestion notification (ECN). Even though Internet routers are increasingly supporting ECN for UDP traffic [17], end-points are required to enforce ECN decisions at the application layer, i.e. through RTP/RTCP protocols [24]. Nevertheless, at the time of this writing, the implementation of RTP/RTCP used by GCC does not support the ECN yet.



(a) SFQ (b) Fq Codel

Figure 8: GCC and TCP flow rates, RTT, and GCC fraction loss dynamics in the case of 99 concurrent TCP flows

6. CONCLUSIONS

In this paper, we have evaluated the interaction between the end-to-end delay-based congestion control algorithm proposed by Google (GCC) with delay-based AQM algorithms, namely CoDel and PIE as well as with flow queuing schedulers such as SFQ and Fq_CoDel. Our analysis has shown that, if only real-time video traffic is considered, the end-to-end congestion control is able to contain the queuing delay with zero losses. On the other hand, PIE and CoDel provide the same queuing delay of DropTail but with the drawback of introducing packet losses. When concurrent TCP traffic is considered, both PIE and CoDel are able to effectively reduce the queuing delays compared to DropTail, but they provoke packet losses on the video flow that increase with the number of TCP flows. These findings are in accordance with those presented in [2] where the author concludes that enforcing tight queuing delays using AQMs at the expense of higher losses is unlikely to be the optimal strategy to improve applications performance. Moreover, we show that flow queuing schedulers offer a better solution since they provide flow isolation. The best interplay is obtained with SFQ that obtains the best performance in terms of queuing delay and packet losses. This result is in agreement with the theoretical analysis presented in [1] in which authors conclude that per flow queuing is the only traffic control needed to satisfy performance requirements.

7. ACKNOWLEDGEMENTS

This work has been partially supported by the projects Platform for Innovative services in the Future Internet (PLATINO - PON01 01007) funded by Italian Ministry of Education, Universities and Research (MIUR), the Google Faculty Award entitled "Congestion Control for WebRTC" and the Future in Research project no. ACYBEH5 funded by the Apulia Region, Italy.

8. REFERENCES

[1] Thomas Bonald and James W Roberts. Internet and the erlang formula. *ACM SIGCOMM Computer Communication Review*, 42(1):23–30, Jan. 2012. doi:10.1145/2096149.2096153.

[2] Bob Briscoe. Insights from curvy red (random early detection). Technical report, TR-TUB8-2015-003, BT, Jul. 2015.

[3] Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl. Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Comm. Surveys & Tutorials*, PP(99):1–1, 2014. doi:10.1109/COMST.2014.2375213.

[4] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio 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. doi:10.1145/2910017.2910605.

[5] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Modelling and Control for Web Real-Time Communication. In *Proc. of IEEE Conference on Decision and Control*, December 2014. doi:10.1109/CDC.2014.7040461.

[6] Markus Fiedler, Tobias Hossfeld, and Phuoc Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2):36–41, 2010. doi:10.1109/MNET.2010.5430142.

[7] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993. doi:10.1109/90.251892.

[8] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Communications of the ACM*, 55(1):57–65, January 2012. doi:10.1145/2063166.2071893.

[9] Y Gong, D Rossi, Claudio Testa, S Valenti, and MD Täht. Fighting the bufferbloat: on the coexistence of aqm and low priority congestion control. *Computer Networks*, 65:255–267, 2014. doi:10.1016/j.bjp.2014.01.009.

[10] Gongbing Hong, James Martin, and James M. Westall. On fairness and application performance of active queue management in broadband cable networks. *Computer Networks*, 91:390 – 406, 2015. doi:10.1016/j.comnet.2015.08.018.

[11] T. Hoiland-Jorgensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. Flowqueue-codel. *IETF Draft*, Oct. 2015.

[12] Toke Hoiland-Jrogensen, Per Hurtig, and Anna Brunstrom. The good, the bad and the wifi: Modern aqms in a residential setting. *Computer Networks*, 89:90 – 106, 2015. doi:10.1016/j.comnet.2015.07.014.

[13] N. Iya, N. Kuhn, F. Verdicchio, and G. Fairhurst. Analyzing the impact of bufferbloat on latency-sensitive applications. In *Proc. of IEEE ICC 2015*, pages 6098–6103, Jun. 2015. doi:10.1109/ICC.2015.7249294.

[14] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling bufferbloat in 3G/4G networks. In *Proc. of ACM IMC*, pages 329–342, 2012. doi:10.1145/2398776.2398810.

[15] N. Khademi, D. Ros, and M. Welzl. The New AQM Kids on the Block: An Experimental Evaluation of CoDel and PIE. In *Proc. of IEEE INFOCOM Workshop on Global Internet Symposium*, pages 85–90, April 2014. doi:10.1109/INFCOMW.2014.6849173.

[16] Paul E McKenney. Stochastic fairness queueing. In *Proc. of IEEE INFOCOM*, pages 733–740, 1990. doi:10.1109/INFCOM.1990.91316.

[17] Stephen McQuistin and Colin S. Perkins. Is explicit congestion notification usable with udp? In *Proc. of ACM IMC*, pages 63–69, Oct. 2015. doi:10.1145/2815675.2815716.

[18] Ali Munir, Ghufran Baig, Syed M. Irteza, Ihsan A. Qazi, Alex X. Liu, and Fahad R. Dogar. Friends, not foes: Synthesizing existing transport strategies for data center networks. In *Proc. of ACM SIGCOMM*, pages 491–502, Aug. 2014. doi:10.1145/2619239.2626305.

[19] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, Jul. 2012. doi:10.1145/2209249.2209264.

[20] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *Proc. of the IEEE International Conference on High Performance Switching and Routing (HPSR)*, Jul. 2013. doi:10.1109/HPSR.2013.6602305.

[21] K Ramakrishnan, Sally Floyd, David Black, et al. The addition of explicit congestion notification (ECN) to IP. *RFC 3168*, 2001. doi:10.17487/rfc3168.

[22] Z. Sarker, V. Singh, X. Zhu, and Ramalho M. Test cases for evaluating rmcat proposals. *IETF Draft*, August 2015.

[23] Stanislav Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. Low extra delay background transport (ledbat). *RFC 6817*, December 2012. doi:10.17487/rfc6817.

[24] Magnus Westerlund, Colin Perkins, Ken Carlberg, and Ingemar Johansson. Explicit congestion notification (ecn) for rtp over udp. *RFC 6679*, Aug. 2012. doi:10.17487/rfc6679.