Making Google Congestion Control robust over Wi-Fi networks using packet grouping

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

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

Stefan Holmer
Google, Sweden
holmer@google.com

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

ABSTRACT

Google congestion control (GCC) has been proposed for the case of delay sensitive traffic (i.e. video-conference) in the WebRTC framework. In this paper we analyze the effect of wireless channel outages on the GCC. We have observed that, when a channel outage ends, there are packets that arrive at the receiver as a burst. This behavior impairs the delay-based controller employed by GCC, resulting in throughput degradation. We propose a solution to make GCC robust with respect to channel outages. In particular, by grouping packets that arrive in a burst, the delay-based controller avoids to misinterpret a burst as network congestion. In order to prove the effectiveness of the proposed solution we have carried out a trace-driven experimental evaluation in a loaded Wi-Fi scenario.

CCS Concepts

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

Keywords

WebRTC, congestion control, Wi-Fi

1. INTRODUCTION

Wireless networks have become an increasingly popular mode of Internet access today due to the diffusion of mobile devices. At the same time, mobile devices have enough processing resources to support high quality real-time video communication. As a consequence, seamless media communication has become commonplace today and popular applications, such as Facebook Messenger and Whatsapp, are implementing such services on their platforms. Two IETF working groups, the RTCWeb and the RMCAT, are standardizing a set of protocols to allow real-time communication among users through Web browsers. In particular, the aim of the RMCAT working group is to standardize a congestion control algorithm designed to deliver real-time audio/video flows. Such algorithms are designed with the goal of minimizing queuing delays and maximizing the throughput in order to respectively enhance interactivity and provide high media quality.

In this context, several congestion control algorithms aiming at improving performance over mobile and Wi-Fi networks have been recently proposed [10, 11, 3, 8]. The effects of wireless link-level mechanisms on end-to-end transport protocols have been well studied in the literature [5] and they can be summarized in terms of rapid variation of the link capacity and variable channel outage periods [10]. In this paper we focus on the issue of the channel outages which remarkably affect the time required to deliver packets. The investigation of the causes of wireless channel outages (i.e. time slotting, lower layer retransmissions) are out of the scope of the paper. The goal here is to analyze the effect of channel outages on the delay-based controller of the Google Congestion Control (GCC) [1] algorithm proposed in the WebRTC framework which runs at the application layer.

The Google Congestion Control (GCC) employs a hybrid loss-based/delay-based and it is particularly interesting since it is already deployed on Google Chrome which is at the time of writing the most popular browser implementing the WebRTC stack[1]. Based on experimental results conducted in a loaded Wi-Fi environment we have measured that channel outages were misinterpreted by the Google Congestion Control as congestion events resulting in sending rate reduction and overall throughput degradation. Moreover, we have observed that when a channel outage ends, all the packets queued in the buffer arrive at the receiver in a burst. To cope with this issue we propose an approach to filter out the detrimental effect of channel outages. By filtering out channel outages events, we have obtained an improvement in terms of throughput without increasing the end-to-end latency.

The reminder of the paper is organized as follows: Section 2 reviews the literature related to recent congestion control for low-delay communication specifically optimized for wireless networks; Section 3 briefly describes the GCC algorithm; Section 4 explains how the channel outages

ANRW '16, July 16 2016, Berlin, Germany
© 2016 ACM. ISBN 978-1-4503-4443-2/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2959424.2959431

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

impair GCC performance showing the results of trace-driven experiments collected in a loaded Wi-Fi environment; Section 5 presents the proposed solution which is tested and evaluated with the same Wi-Fi traces and Section 6 concludes the paper.

2. RELATED WORK

It is well know that TCP performance are remarkably affected in scenarios characterized by lossy links or high bandwidth-delay product. A comprehensive survey of TCP limitations is provided by [3] which also proposes a utility-based algorithm to overcome such limitations. However, the choice of the utility function is still considered an open issue in [3]. TCP congestion control is not able to efficiently handle rapid variation of link capacity or channel outages [3] which are typical of wirelesses links. Additionally, TCP is also prone to the bufferbloat phenomenon [6] due to its probing mechanism originating periodic cycles during which network queues are first filled and then drained. These queue oscillations provoke a time-varying stochastic delay component that adds to the propagation time and make time-sensitive communications problematic. Consequently, delay sensitive applications typically do not employ TCP, but favor the implementation of congestion control on top of UDP as in the case of WebRTC [1].

Sprout [10] has reawakened the interest of scientific community towards the design of congestion control algorithms for interactive applications over wireless network in order to address the aforementioned TCP limitations. Sprout employs a stochastic-based algorithm which explicitly computes the sending rate to contain delays while maximizing the throughput; it has been experimentally evaluated in a wireless emulated environment showing that it provides improvements compared to Skype, Facetime, and Hangout in a single flow scenario. However, intra and inter-protol fairness of Sprout has not been investigated. Moreover since it relies on a model to compute the sending rate, uncertainty in the model may degrade performance [8].

Verus [11] shares the same goal with Sprout. The algorithm employs delay gradient measurements to detect congestion and computes the sending window. Authors have performed a performance comparison with Sprout over 3G and 4G networks, showing that Versus generally performs similarly to Sprout in terms of induced delay with slightly higher throughput.

CQIC [8] proposes a cross-layer congestion control which exploits physical layer information exchanged between nodes of wireless network to predict the capacity of the underlying layer. The algorithm has been implemented in the Google QUIC framework [2] showing that, when the bottleneck is the wireless link, CQIC outperforms TCP CUBIC in terms of throughput and latency.

The closest manuscript to ours is Rebera [7] which is a congestion control algorithm designed for sending real-time video content with the aim of maximizing the video transmission rate while keeping the queuing delays as low as possible. Authors deal with burst arrivals by discarding these measurements if packets inter-arrival time is lower than 10ms.

In this work we propose an approach to deal with channel outages and video-frame burst arrivals. We have implemented such solution in GCC, a congestion control algorithm today employed in the Google Chrome WebRTC framework and Google Hangouts to transport video conferencing traffic on top of RTP/UDP protocols. It has already been shown that GCC is able to contain queuing delays and at the same adapts its sending rate to match the link capacity providing intra and inter-protocol fairness [1]. At the best of our knowledge this is the first work that analyzes the behavior of GCC over wireless links.

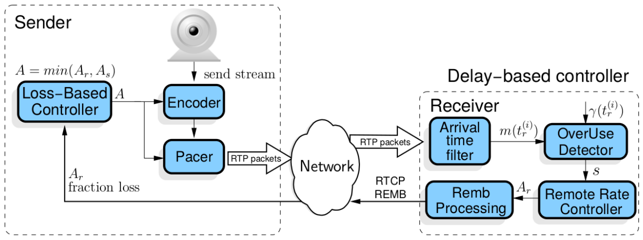


Figure 1: Google congestion control architecture

3. GOOGLE CONGESTION CONTROL

This Section briefly 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.

Figure 1 shows the architecture of the end-to-end GCC algorithm [1]. The sender employs a UDP socket to send RTP packets and receive RTCP feedback reports from the receiver. The algorithm has two components: (i) 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; (ii) a *loss-based* controller, placed at the sender, that computes a rate A_s and it sets the target sending bitrate A equal to $\min(A_r,A_s)$.

Sending rate actuation. Before starting to provide details on the two controllers we briefly describe how the target bitrate is actuated by the sending engine. Figure 1 shows that the sender sets the target sending bitrate A equal to the minimum between A_r and A_s. The target bitrate A is fed both to the Pacer and to the Encoder. The Encoder strives to produce a bitrate as close to this target as possible. When a video frame is produced it is fed into a Pacer queue. The Pacer divides the video frame into *groups of packets* and sends them to the network every ΔT which in our case is set equal to 5ms. The delay-based controller at the receiver operates every time a group of packets arrives. Pacing is used by rate-based algorithms to actuate the sending rate computed by the congestion control. Moreover, pacing helps the network to better absorb packets as opposed to the case of using a window-based sender which would send data bursts.

The sender-side congestion control. It is a *loss-based* congestion control algorithm that acts every time $t_r^{(k)}$ the k-th RTCP report message arrives at the sender or every time $t_r^{(k)}$ the k-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

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

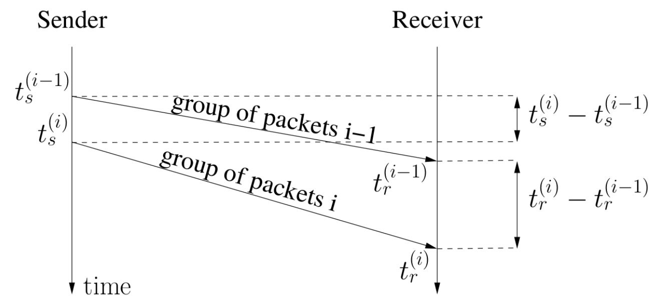


Figure 2: One way delay variation measurement

$f_l(t_r^{(k)})$ computed as described in [9]. Based on $f_l(t_r^{(k)})$, the controller computes the rate $A_s(t_r^{(k)})$, measured in kbps, according to the following equation:

$$A_s(t_r^{(k)}) = \begin{cases} A_s(t_r^{(k-1)})(1 - 0.5 f_l(t_r^{(k)})) & f_l(t_r^{(k)}) > 0.1 \\ 1.05(A_s(t_r^{(k-1)})) & f_l(t_r^{(k)}) < 0.02 \\ A_s(t_r^{(k-1)}) & \text{otherwise} \end{cases} \tag{1}$$

The rationale of (1) is simple: (i) when the fraction of lost packets is considered small ($0.02 \leq f_l(t_r^{(k)}) \leq 0.1$), A_s is kept constant, (ii) if a high fraction lost is estimated ($f_l(t_r^{(k)}) > 0.1$) the rate is multiplicatively decreased (3) when the fraction lost is considered negligible ($f_l(t_r^{(k)}) < 0.02$), the rate is multiplicatively increased. After A_s is computed through (1), the target bitrate is set as $A \leftarrow \min(A_s, A_r)$ 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 $t_r^{(i)}$ the i-th *group of packets* is received, the *one way queuing delay variation* $m(t_r^{(i)})$ is estimated by the *arrival-time filter* (ATF). The goal of this block is to produce an estimate $m(t_r^{(i)})$ of the one way delay variation. For this purpose, we employ a Kalman filter [1] that estimates $m(t_r^{(i)})$ based on the *measured one way delay variation* $d_m(t_r^{(i)})$ which is computed as follows (see Figure 2):

$$d_m(t_r^{(i)}) = (t_s^{(i)} - t_s^{(i-1)}) - (t_r^{(i)} - t_r^{(i-1)}) \tag{2}$$

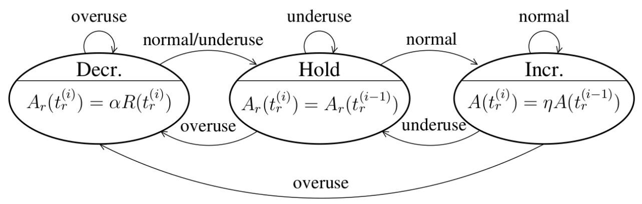
where $t_s^{(i)}$ is the time at which the first packet of the i-th group has been sent and $t_r^{(i)}$ is the time at which the last packet of the group has been received.

The one way delay variation is considered as the sum of two components [1]: (i) the *one way queuing time variation* $m(t_r^{(i)})$, and (ii) the *network jitter* $n(t_r^{(i)})$ modeled as Gaussian noise. The following mathematical model of the one way delay variation is assumed [1]:

$$d(t_r^{(i)}) = m(t_r^{(i)}) + n(t_r^{(i)}) \tag{3}$$

the one way queuing time variation $m(t_r^{(i)})$ accounts also for the contribution of the transmission time variation due to the different size of consecutive *groups of packtes*, since it can be considered negligible with respect to the queuing delay variation.

Then, the *over-use detector* compares the estimated one way queuing delay variation $m(t_r^{(i)})$ with an adaptive threshold $\gamma(t_r^{(i)})$ proposed in [1]: when $m(t_r^{(i)})$ gets above $\gamma(t_r^{(i)})$, the network is considered congested and the *overuse* signal is generated; on the other hand, if $m(t_r^{(i)})$ decreases below $-\gamma(t_r^{(i)})$, the network is considered underused and the *underuse* signal is generated; when $m(t_r^{(i)})$ falls back in $[-\gamma(t_r^{(i)}), \gamma(t_r^{(i)})]$ a *normal* signal is produced.



overuse
underuse
normal
normal/underuse
normal
Decr.
$A_r(t_r^{(i)}) = \alpha R(t_r^{(i)})$
Hold
$A_r(t_r^{(i)}) = A_r(t_r^{(i-1)})$
Incr.
$A(t_r^{(i)}) = \eta A(t_r^{(i-1)})$
overuse
underuse
overuse

Figure 3: Remote rate controller

Finally, the signal s is fed to the *remote rate controller* which drives the finite state machine (FSM) shown in Figure 3 whose goal is to empty the queues along the end-to-end path. The rationale is the following: when the bottleneck buffers start to build-up, the estimated one way delay variation $m(t_r^{(i)})$ becomes positive. The overuse 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 that the estimated one way delay variation $m(t_r^{(i)})$ 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 is emptied. When this occurs, $m(t_r^{(i)})$ approaches 0 and the overuse detector generates a normal signal, which drives the machine into the "increase" state. A_r is increased, decreased or kept constant depending on its state. In particular A_r is set according to the equations shown in the states of Figure 3, where $\eta \in [1.005, 1.3]$, $\alpha \in [0.8, 0.95]$, and $R(t_r^{(i)})$ is the receiving rate measured in the last 500ms. It is worth noticing that A_r cannot exceed $1.5R(t_r^{(i)})$. The computed rate A_r is sent to the sender through REMB messages.

4. PERFORMANCE ISSUES OVER WI-FI NETWORKS

In this Section we illustrate the performance issues of the algorithm described in Section 3. We focus on the impact of Wi-Fi channel outages on the delay-based controller. The delay-based algorithm dominates the control action since it strives to react before a loss is induced.

4.1 Problem statement

Figure 4 shows the effect of channel outages on the arrival time $t_r^{(i)}$ of groups of packets. The goal is to analyze the effect of such an issue on the delay-based controller. For the sake of simplicity Figure 4 shows that a video frame fits exactly in one group of packets. During channel outages packets are queued up in the network buffers and when the outage ends they arrive at the received in a burst. The delay-based controller at the receiver measures the *one way delay variation* $d_m(t_r^{(i)})$ according to (2). From Figure 4 we can distinguish three different patterns of d_m: (i) in the absence of channel outages $d_m(t_r^{(2)}) = t_r^{(2)} - t_r^{(1)} - (t_s^{(2)} - t_s^{(1)})$

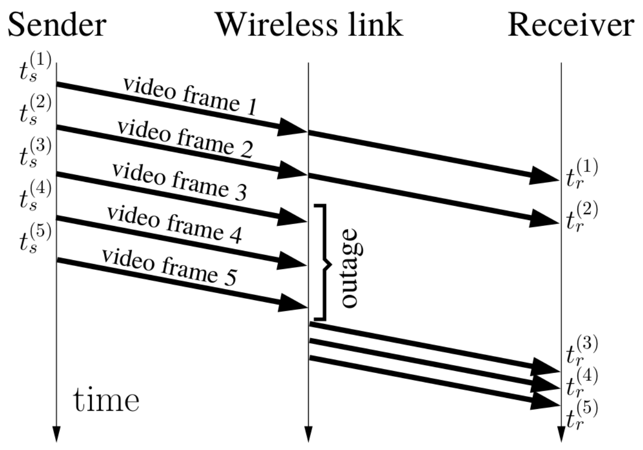


Figure 4: Effect of a wireless channel outage on the arrival time of groups of packets (in this example a video frames fits in one group of packets)

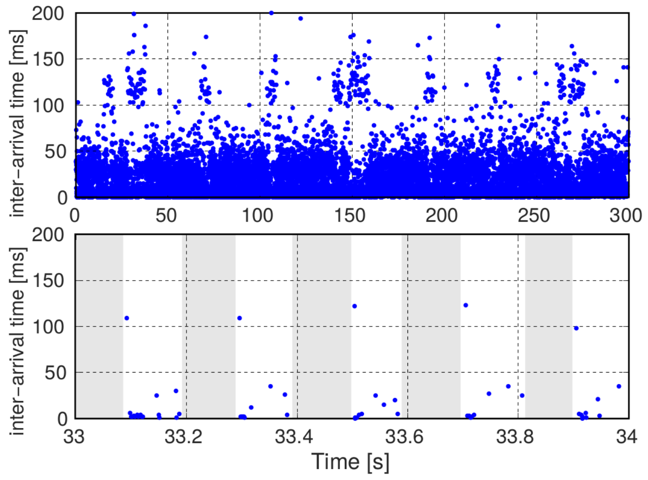


Figure 5: Groups of packets inter-arrival time and temporal zoom in the time interval $[33, 34]$**s. Channel outages are highlighted in grey**

is roughly equal to 0; (ii) right after the channel outage $d_m(t_r^{(3)}) = t_r^{(3)} - t_r^{(2)} - (t_s^{(3)} - t_s^{(2)})$ is large and positive and (iii) when the outage ends $d_m(t_r^{(4)}) = t_r^{(4)} - t_r^{(3)} - (t_s^{(4)} - t_s^{(3)})$ is negative and roughly equal to the opposite of the inter-departure time $(t_s^{(i)} - t_s^{(i-1)})$, since we can consider $(t_r^{(i)} - t_r^{(i-1)}) \simeq 0$, i.e. the inter-arrival time during the burst arrivals is negligible. These patterns might be interpreted as congestion events by the delay-based controller resulting in throughput degradation. In Section 5 we propose a solution to this issue.

4.2 Experimental Validation

To reproduce the issue presented in Section 4.1, we have collected traces over a high loaded 802.11n Wi-Fi network which was shared among several users. Traces have been collected sending 1200byte large packets at a constant bitrate equal to 3Mbps. The traces contain the time at which the Wi-Fi network has forwarded the packet to the receiver and are available on-line[3] in the open-source Chromium Simulation Framework. We reproduce a video conference session delivering the RTP packets at the time reported in the trace. We have employed a video encoder which generates 30 video frames per second which can produce a bitrate in the range $[50, 2500]$kbps based on the value requested by the congestion control algorithm.

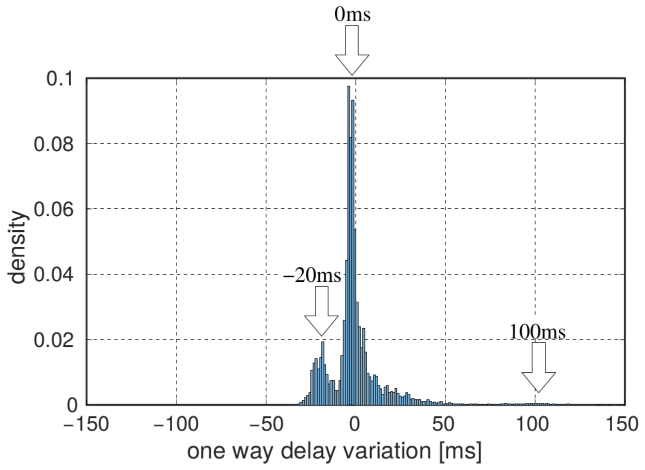


Figure 6: Density function of the measured one way delay variation among group of packets on a loaded Wi-Fi network

Figure 5 shows the inter-arrival time of the groups of packets, that have been sent through Wi-Fi network, measured at the receiver. Figure 5 (a) shows the measurements in the first 100 seconds of a video call, whereas Figure 5 (b) shows a temporal zoom in the interval $[33, 34]$s. The inter-arrival time of groups of packets experiences a high variance due to the effect of outages. Figure 5 (b) shows in grey the channel outages which are time-varying and are followed by burst of arrivals. This pattern repeats throughout all the video call.

To better understand the effect of channel outages on the one way delay variation d_m, Figure 6 shows the probability density function of d_m for the experiment of Figure 5. The density function can be approximated as the superposition of three Gaussian-like distributions centered around to -20 ms, 0 ms, and 100 ms. This confirms the expected patterns of d_m described in Section 4.1. The distribution with mean equal to 0 accounts for pattern case (i) in which groups of packets are delivered without delay variations. The distribution centered at 100ms accounts for pattern case (ii). Finally, the distribution centered at -20ms account for pattern case (iii) due to the burst; as we stated in Section 4.1, when burst of arrivals occur, d_m is negative and roughly equal to the opposite of the inter-departure time $(t_s^{(i)} - t_s^{(i-1)})$. In our case the inter-departure time is determined by two factors: (i) the time interval of the Pacer $\Delta T = 5$ ms and (ii) the time interval at which video frame are generated which is 33ms (30fps). These two factors originates to two density functions: (i) one with mean around 5ms which is not distinguishable from the one with mean equal to 0ms and (ii) one with mean roughly equal to -20ms which is determined by the difference between the departure time

[3]https://chromium.googlesource.com/external/webrtc/+/master/webrtc/modules/remote_bitrate_estimator

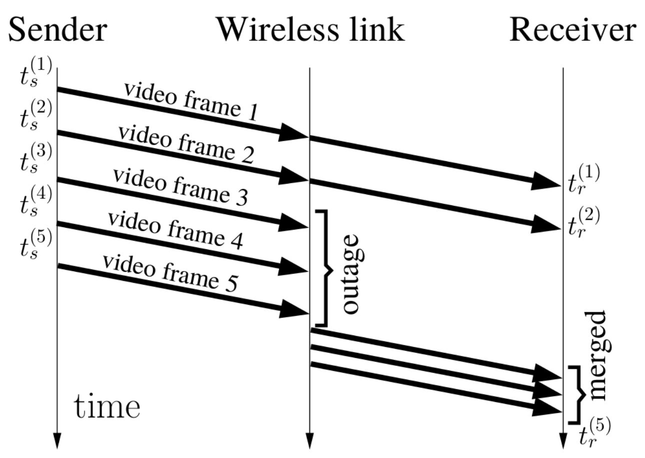


Figure 7: Application of Algorithm 1 in the case depicted in Figure 4

of the last group of packets of a video frame and the first group of packets of the next video frame. All of these density functions are affected by jitter noise modeled as $n(t_r^{(i)})$ in (3) which accounts for their variance.

Based on this analysis we conclude that the density function of d_m needs to be pre-filtered before feeding the arrival time filter.

5. PROPOSED SOLUTION

In this Section we propose a pre-filtering mechanism to be placed before the ATF block of Figure 1.

5.1 Pre-filtering

In order to filter out the effect of the channel outages on the density function of d_m, the pre-filtering mechanism merges groups of packets that arrive in a burst. Algorithm 1 shows the pseudo-code of the pre-filtering.

Algorithm 1 operates at the receiver every time an RTP packet arrives. It decides if the arriving packets should be merged in one group of not. First it verifies if packets belong to a burst arrival. To this purpose it checks if the inter-arrival time $(t_r^{(i)} - t_r^{(i-1)})$ is smaller than ΔT and if the measured one way delay variation $d_m(t_r^{(i)}) = (t_s^{(i)} - t_s^{(i-1)}) - (t_r^{(i)} - t_r^{(i-1)})$ is negative. In this case it assumes that packets belong to a burst arrival. If it is not the case Algorithm 1 verifies if a new group is arriving by checking their departure time. If inter-departure time $(t_s^{(i)} - t_s^{(i-1)})$ is greater than or equal to ΔT (the minimum inter-departure time among groups), a new group is arriving; at this point the ATF can be updated, using the arrival and the departure time of the previous group.

In order to clarify how Algorithm 1 works, Figure 7 shows the same example of Figure 4 when Algorithm 1 is used; in this case we measure only 2 samples for d_m instead of 4. In particular we measure: (i) $d_m(t_r^{(2)}) = t_r^{(2)} - t_r^{(1)} - (t_s^{(2)} - t_s^{(1)})$ and (ii) $d_m(t_r^{(5)}) = t_r^{(5)} - t_r^{(2)} - (t_s^{(5)} - t_s^{(2)})$ which accounts the arrival and departure time of the merged group $(t_r^{(5)},\ t_s^{(5)})$ and the arrival and departure time of the last received the group $(t_r^{(2)},\ t_s^{(2)})$. As a consequence the filter ATF will be fed with only 2 samples.

```
$(t_r, t_s) \leftarrow$ on RTP packet arrival;
if $((t_r - t_{r\_old}) < \Delta T)$ and $(d_m(t_r) < 0)$ then
    Merge packet in the current Group;
else
    if $(t_s - t_{s\_old}) \geq \Delta T$ then
        New Group is arriving;
        Update ATF with $(t_{r\_old}, t_{s\_old})$;
    else
        Merge packet in the current Group;
$t_{r\_old} \leftarrow t_r$;
$t_{s\_old} \leftarrow t_s$;
```

Algorithm 1: Pre-filtering approach

5.2 Validation and comparison

To validate the solution and prove its effectiveness, in this Section we have carry out a trace-driven comparison using the Wi-Fi traces described in Section 4.1.

First of all Figure 8 shows the PDF of the one way delay variation d_m when pre-filtering is used. We observe that, when Algorithm 1 is used, the two distributions respectively centered at -20ms and 100ms have been filtered out, obtaining only one distribution centered at 0ms. This signal will be fed to arrival time filter ATF shown in Figure 1.

In order to quantitatively assess the effectiveness of this solution we carry out a comparison considering QoS metrics such as average sending bitrate and one way delay, which are known to be well correlated with video QoE metrics [4]. In particular, we consider: (i) *Throughput*, computed as the sending rate of the video flow and (ii) *One Way Delay*, measured as the difference between the departure time of a group of packets and its arrival time. Then we compare the results with respect to the case in which pre-filtering is not used.

Figure 9 shows a comparison between the dynamics of the throughput and the one way delay of a GCC flow over wireless link when pre-filtering is used (b) and when is not used (a). Figure 9 clearly shows that, with pre-filtering, the average sending rate is higher without remarkably worsen the one way delay dynamics. In order to get a better insight from this comparison, Figure 10 shows the average sending

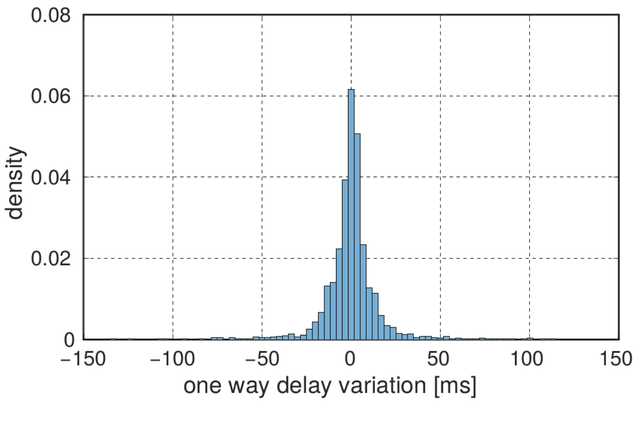
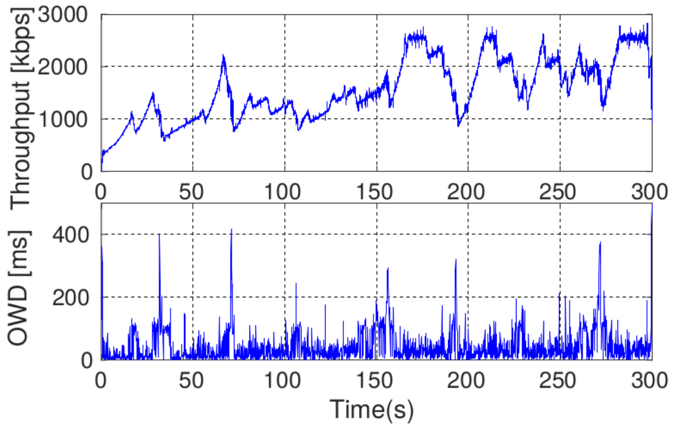
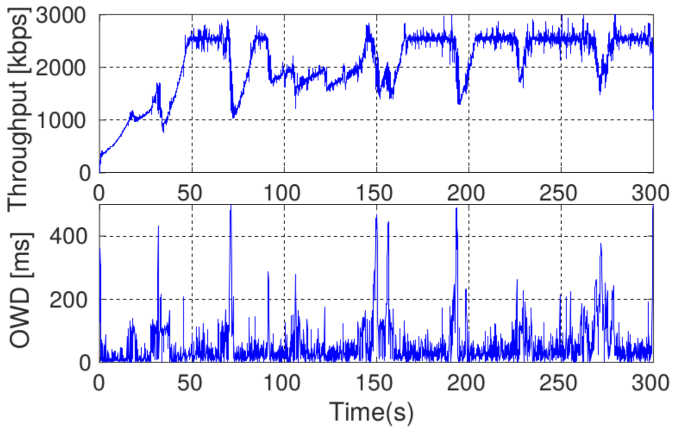


Figure 8: Density function of the measured one way delay variation among group of packets on a loaded Wi-Fi network when pre-filtering is applied



(a) w/o pre-filtering



(b) w/ pre-filtering

Figure 9: Throughput and one way delay dynamics of a GCC flow over wireless link with (a) and without (b) pre-filtering

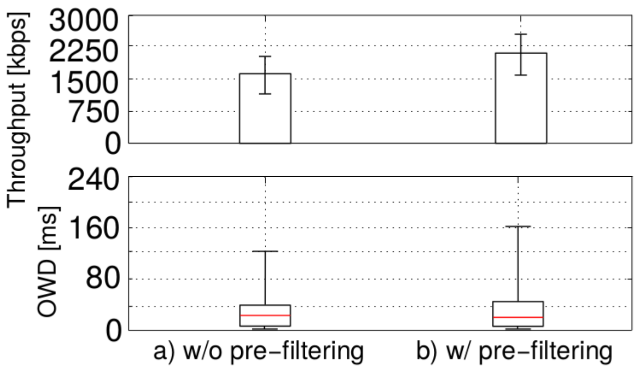


Figure 10: Average throughput with standard deviation and one way delay percentiles with (a) and without (b) pre-filtering

rate with the standard deviation and the one way delay using a box and whisker plot: the bottom and top of the box are respectively the 25-th and 75-th percentile, whereas the red band in the box is the median; the end of the whiskers represent the 5-th and 95-th percentile. Overall we observe more than 20% improvement of the average value of the sending bitrate; the standard deviation is roughly equal in both cases. In terms of delay we observe that only the 95th percentile is higher in the case pre-filtering is used.

6. CONCLUSIONS

In this paper we have investigated the effect of wireless channel outages on the Google Congestion Control used in the WebRTC framework. In particular we have carried out a trace-driven evaluation employing traces collected in a loaded Wi-Fi network. We have observed that the channel outages might lead to throughput degradation due to the fact that delay-based controller interprets these events as congestion signal. To this purpose we have designed a pre-filtering mechanism which, by merging packets that arrive as a burst, provides roughly 20% improvement of the average throughput without remarkably increasing the queuing delay.

7. ACKNOWLEDGMENTS

This work has been partially supported by 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] 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.

[2] G. Carlucci, L. De Cicco, and S. Mascolo. Http over udp: an experimental investigation of quic. In *Proc. of 30th ACM/SIGAPP Symposium On Applied Computing (SAC 2015)*, Salamanca, Spain, April 2015.

[3] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proc. of USENIX NSDI*, pages 395–408, Oakland, CA, May 2015.

[4] M. Fiedler, T. Hossfeld, and P. Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2):36–41, 2010.

[5] A. Gurtov and S. Floyd. Modeling wireless links for transport protocols. *ACM SIGCOMM CCR*, 34(2):85–96, Apr. 2004.

[6] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. In *Proc. of ACM IMC*, pages 329–342, 2012.

[7] E. Kurdoglu, Y. Liu, Y. Wang, Y. Shi, C. Gu, and J. Lyu. Real-time Bandwidth Prediction and Rate Adaptation for Video Calls over Cellular Networks. In *Proc. of the ACM Multimedia Systems Conference*, Klagenfurt, Austria, May 2016.

[8] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *Proc. of the HotMobile, Workshop on Mobile Computing Systems and Applications*, pages 45–50, Feb. 2015.

[9] H. Schulzrinne, S. Casner, S. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *RFC 3550*, 2003.

[10] 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.

[11] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive congestion control for unpredictable cellular networks. In *Proc. of ACM SIGCOMM*, volume 45, pages 509–522, Aug. 2015.