Congestion Control for Real-time Communications: a comparison between NADA and GCC

Gaetano Carlucci, Luca De Cicco, *Member, IEEE,* Cesar Ilharco, and Saverio Mascolo, *Senior Member, IEEE*

***Abstract*— Congestion control for Web real-time communication (WebRTC) is a hot topic currently addressed at the IETF. Differently from congestion control for TCP, congestion control for WebRTC not only aims at containing packet losses, but also aims at minimizing queuing delays to provide interactivity. In this paper we describe two algorithms under discussion at IETF: *Network Assisted Dynamic Adaptation* (NADA) proposed by Cisco Systems and *Google Congestion Control* (GCC) proposed by Google. A performance comparison in a simulation environment is carried out. Results show that GCC exhibits slow convergence whereas NADA exhibits a remarkable oscillating behavior.**

***Index Terms*— low-delay communication, congestion control**

I. Introduction and Related Work

Congestion control is a required functionality for all applications generating flows over the Internet to both avoid network congestion collapse and provide the user with an acceptable perceived quality. Real-time multimedia traffic, such as the one generated by video-conferencing applications, differs significantly from bulk data traffic both in terms of requirements and employed protocols for delivery. On one hand, bulk data traffic is insensitive to packet delivery delay and it essentially requires minimization of the flow completion time [10]. On the other hand, real-time flows are highly affected by connection latency that should be kept as low as possible to provide users with an interactive communication experience.

Bulk data flows use the Transmission Control Protocol (TCP), which implements reliability through retransmissions at the cost of delayed delivery of packets, whereas real-time flows employ the User Datagram Protocol (UDP), which does not implement retransmissions. However, since the UDP does not provide a congestion control algorithm, real-time multimedia applications have to implement this feature at the application layer.

The novelty in the control of real-time flows stems from the fact that, besides congestion avoidance, it is required to also control network queue lengths in order to contain delays. Connection delay is made of two components. A constant *propagation delay*, and a *queueing delay* which depends on the queue lengths. Since connection latency has to be minimized, such algorithms employ delay measurements to detect and react to congestion by changing the sending rate according to the designed control law. Several control approaches have been proposed to design efficient congestion control algorithms for real-time multimedia flows. The first proposed delay-based control algorithms employed the measured Round Trip Time (RTT), i.e. the delay from the sender to the destination and then back to the sender, to infer congestion. In this case the knowledge of the end-to-end RTT characteristics is required to establish delay thresholds used to infer congestion. Algorithms using such an approach are, f.i., TCP Vegas [1] and TCP FAST [23]. 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, which inflates queues in the backward path, or when competing with loss-based flows [11]. 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) [21] and TCP Santa Cruz [17]. It has been shown that such algorithms may be affected by the so called *latecomer effect*: when two flows share the same bottleneck the second flow may starve the first one [6]. Recently the idea of employing the RTT gradient, i.e. the variation of RTT samples, has been proposed to overcome the aforementioned *latecomer effect* with some promising results [12].

Several papers have proposed mathematical models to describe and analyze the dynamics of congestion control algorithms [15], [14], [22], [8], [3]. Issues studied in the literature comprise robust stability of congestion control algorithms in the case of time-varying delays in the control loop [2], [9], fairness properties of AIMD congestion control algorithms [22], interaction of end-to-end congestion control algorithms with Active Queue Management schemes [13], [16].

In this paper we compare two congestion control algorithms designed for real-time multimedia flows and proposed for standardization within the IETF working group (WG) *RTP Media Congestion Avoidance Techniques*[1] (RMCAT): the *Network Assisted Dynamic Adaptation* (NADA) [24], which employs one-way delay measurements to infer congestion, and the *Google Congestion Control* (GCC), which employs one-way delay gradients. The paper provides a performance comparison between the two congestion control algorithms in several scenarios.

II. Algorithms

This Section briefly describes the algorithms involved in our study. Section II-A describes Network-Assisted Dynamic

G. Carlucci, L. De Cicco, and S. Mascolo are with the Dipartimento di Ingegneria Elettrica e dell'Informazione at Politecnico di Bari, Via Orabona 4, 70125, Bari, Italy Emails: gaetano.carlucci@poliba.it, luca.decicco@poliba.it, mascolo@poliba.it. C. Ilharco was at Google Inc. when this work was done. Email: cesar.ilharco@gmail.com.

[1]`http://datatracker.ietf.org/wg/rmcat/`

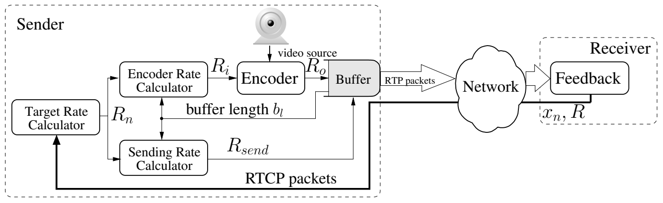

Fig. 1: Network-Assisted Dynamic Adaptation architecture

Adaptation (NADA) algorithm, whereas section II-B revises Google Congestion Control (GCC) which is used in Google Chrome browsers to implement congestion control functionalities in the WebRTC stack and in Google Hangouts.

A. *Network Assisted Dynamic Adaptation (NADA) [24]*

Figure 1 shows the architecture employed by NADA. The sender sends media content in RTP (Real-Time Protocol) [20] packets over UDP. The receiver computes an aggregated congestion signal x_n which is sent to the sender in RTCP (RTP Control Protocol) [20] packets. Based on x_n, the sender adapts its sending rate to match the available bandwidth. The following description is based on the NADA draft [24].

The receiver-side controller: The receiver computes a congestion signal $x_n(t_i)$ every time t_i a RTP packet is received. The aggregated congestion signal $x_n(t_i)$ is made of two terms:

$$x_n(t_i) = \tilde{d}(t_i) + D_{loss} \cdot p_{loss}(t_i) \tag{1}$$

$\tilde{d}(t_i)$ is computed as described in the following: 1) when the i-th packet is received the *one-way delay* is measured as $d(t_i) = t_i - T_i$, where T_i is the time at which the i-th packet has been sent and t_i is the time at which it has been received; 2) the queuing delay $d_q(t_i)$ is then computed by subtracting from $d(t_i)$ the baseline delay $d_{min}(t_i)$ which is the minimum delay measured in the session; 3) $d_q(t_i)$ is first filtered with a median filter (window size 5) and then by using an exponential smoothing $\overline{d}_q(t_i) = (1-\alpha)\cdot d_q(t_i) + \alpha \cdot \overline{d}_q(t_{i-1})$ with $\alpha = 0.9$; 4) $\bar{d}_q(t_i)$ is fed to the non-linear function shown in Figure 2 which returns the value of $\tilde{d}(t_i)$; when $\overline{d}_q(t_i)$ is outside the range $[0, 400]$ms $\tilde{d}(t_i) = 0$.

p_{loss} is the fraction of packet loss measured in a time window of 500ms; D_{loss} is the equivalent delay penalty for a loss which is set to 1 second. Finally, the receiver computes the average received rate $R(t_i)$ in the last time window of 500ms. It reports to the sender the value of $x_n(t_i)$ and $R(t_i)$ every Δ =100ms in RTCP packets.

The sender-side controller: The controller is clocked to the time t_k a feedback packet is received. It operates in two modes: 1) *accelerated ramp up* or 2) *gradual rate update*.

***Accelerated ramp up*:** the sender operates in this mode when the received aggregated congestion signal x_n is less than 10ms. The target rate is multiplicatively increased according to:

$$R_n(t_k) = (1 + \gamma(t_k))R_n(t_{k-1}) \tag{2}$$

where $R_r(t_k)$ is the average received rate computed by the receiver and $\gamma(t_k) = \min(0.2, \overline{Q}/(RTT(t_k) + \Delta)$ where $\overline{Q} = 50$ms (Figure 2).

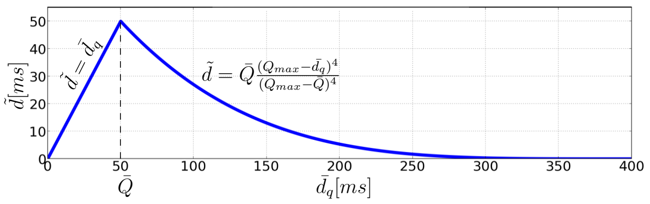

Fig. 2: Non-linear function mapping $\bar{d}_q$ to $\tilde{d}$

Gradual rate update: the target rate changes according to:

$$R_n(t_k) = R_n(t_{k-1}) \cdot (1 - K_1 x_o(t_k) - K_2 x_d(t_k)) \tag{3}$$

where $K_1$and K_2 are two positive constants [24] and x_o is the distance of x_n from a reference value x_{ref}:

$$x_o(t_k) = x_n(t_k) - \frac{R_{max}}{R_n(t_k)} x_{ref} \tag{4}$$

and x_d accounts for x_n variations:

$$x_d(t_k) = x_n(t_k) - x_n(t_{k-1}) \tag{5}$$

$x_o(t_k)$ depends on the maximum flow rate R_{max} and on the reference congestion signal $x_{ref} = 20$ms. Finally the target rate must be contained in the range $[R_{min}, R_{max}]$ based on the video encoder constraints (see Section III).

Encoder Rate Control: the target rate $R_n(t_k)$ requested by the controller may deviate from the rate $R_o(t_k)$ produced by the video encoder. NADA employs a rate shaping buffer to absorb the mismatch between $R_o(t_k)$ and $R_n(t_k)$. In order to keep the buffer as empty as possible so that delayed transmission of packets is avoided, the instantaneous buffer length $b_l(t_k)$ is sent back to the *Encoder Rate Calculator* and to the *Sender Rate Calculator* with two purposes : 1) to deplete the rate shaping buffer faster by increasing the sending rate $R_{send} = R_n + \beta b_l(t_k)$; 2) to limit incoming packets of the rate shaping buffer by reducing the video encoder target rate $R_i(t_k) = R_n(t_k) - \beta b_l(t_k)$, ($\beta = 0.1$).

B. *Google Congestion Control (GCC) [4]*

Figure 3 shows the architecture of the end-to-end Google Congestion Control (GCC) algorithm [4]. Similarly to NADA, the GCC sender employs a UDP socket to send RTP packets and receive RTCP feedback reports from the receiver. The sender employs two controllers: 1) a *delay-based* controller, that computes a rate A_r with the aim of keeping the queuing delay small; 2) a *loss-based* controller, that computes a target sending bitrate A_s that cannot exceed A_r. The receiver sends RTCP feedback packets to the sender.

The receiver-side controller: the receiver acts every time t_i a packet is received. It computes the *one-way delay variation* $d_m(t_i) = t_i - t_{i-1} - (T_i - T_{i-1})$, where T_i is the timestamp at which the i-th packet has been sent and t_i is the timestamp at which it has been received. Moreover it computes the fraction of lost packets $f_l(t_i)$ as described in the RTP RFC [20]. It sends to the sender the value of $d_m(t_i)$, $f_l(t_i)$, and $R(t_k)$ which is the average received rate in the last 500ms.

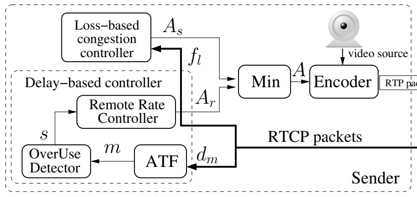

Fig. 3: Google Congestion Control architecture

The sender-side controller: the sender side acts every time t_k a RTCP report message arrives at the sender. It employs two controllers: 1) a *delay-based* controller and 2) a *loss-based* controller.

The loss-based controller: Based on $f_l(t_k)$, it computes the rate $A_s(t_k)$ according to the following equation:

$$A_s(t_k) = \begin{cases} A_s(t_{k-1}) \cdot (1 - 0.5 f_l(t_k)) & f_l(t_k) > 0.1 \\ 1.05 \cdot A_s(t_{k-1}) & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases} \tag{6}$$

The rationale of (6) 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.

The delay-based controller: this controller is made of the three components shown in Figure 3. The value of the measured *one-way delay variation* $d_m(t_k)$ is fed to the *arrival-time filter* (ATF) which estimates the *one-way queuing delay* $m(t_k)$. The *one-way delay variation* $d(t_k)$ is modeled as the sum of three components [4]: 1) the *transmission time variation*, 2) the *one-way queuing time variation* $m(t_k)$, and 3) the *network jitter* $n(t_k)$. The following mathematical model of the one way delay variation is assumed [4]:

$$d(t_k) = \frac{\Delta L(t_k)}{C(t_k)} + m(t_k) + n(t_k) \tag{7}$$

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

Then, the *over-use detector* compares the estimated one-way queuing delay variation $m(t_k)$ with an adaptive threshold $\gamma(t_k)$ proposed in [5], since it has been shown that with a static threshold [7] the algorithm has fairness issues. In particular when $m(t_k)$ gets above $\gamma(t_k)$, the network is considered congested and the *overuse* signal is generated; on the other hand, if $m(t_k)$ decreases below $-\gamma(t_k)$, the network is considered underused and the *underuse* signal is generated; when $m(t_k)$ falls back in $[-\gamma(t_k), \gamma(t_k)]$ 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 4 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 4, where $\eta \in [1.005, 1.3]$, $\alpha \in [0.8, 0.95]$, and $R(t_k)$ is the receiving rate measured in the last 500ms by the receiver. It is worth noticing that A_r cannot exceed $1.5R(t_k)$.

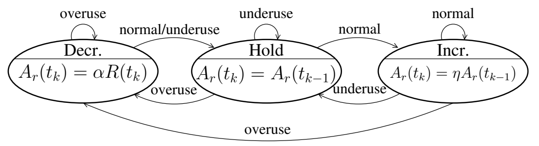
overuse
normal/underuse
underuse
normal
normal
Decr.
$A_r(t_k) = \alpha R(t_k)$
Hold
$A_r(t_k) = A_r(t_{k-1})$
Incr.
$A_r(t_k) = \eta A_r(t_{k-1})$
overuse
underuse
overuse

Fig. 4: GCC Remote rate controller

After A_s and A_r are computed, the target bitrate A it is set as $A \leftarrow \min(A_s, A_r)$ to avoid that A_s exceeds the value of A_r. As in the case of NADA, the target bitrate A must be contained in the range $[R_{min}, R_{max}]$ based on the video encoder constraints (see Section III).

III. Testbed and Metrics

In this section we describe the topology of the scenario and the simulation tool employed to carry out the performance evaluation and comparison.

The simulator. GCC and NADA algorithms have been implemented in the open-source Chromium Simulation Framework and are available on-line[2]. Figure 5 shows an essential view of the topology employed in the simulations. Two nodes (Node 1 and Node 2) are connected through a link whose parameters can be configured by the framework APIs. Each node runs a configurable number of NADA, GCC and TCP sources or sinks. The one-way propagation delay has been set to 50ms both on the direct path and on the reverse path in all of the simulations, resulting in a round trip propagation delay RTT_{min} equal to 100ms. Moreover the maximum bottleneck queue size $\bar{T}_q$ has been set to 300ms in all of the simulations. Finally, when packets are sent in the network a variable delay is applied in order to emulate the network jitter noise. The jitter noise has been chosen with a truncated 3σ Gaussian distribution in the range $[0, 15]$ms ($\sigma = 5$ms) [19].

Video Encoder and TCP settings. The TCP sources employ the NewReno congestion control. The video encoder generates 30 video frames per second and their size is adapted based on the rate computed by the congestion control algorithm. The encoder produces a bit in the range $[50, 2500]$kbps such that the parameter R_{min} and R_{max} for both NADA and GCC are respectively 50kbps and 2500kbps.

Metrics. We consider the following metrics to evaluate the performance of the video flows controlled by both NADA and GCC: 1) *Channel Utilization* $U = R/b$, where b is the known link capacity and R is the average received rate; for every simulation we measure the average value and the standard deviation; 2) *Loss ratio* $l = $ (byte lost)/(byte sent); we measure the cumulative value; 3) *Queuing delay* measured as the difference between the one-way delay and the

[2]`https://chromium.googlesource.com/external/webrtc/+/master/webrtc/modules/remote_bitrate_estimator/test/`

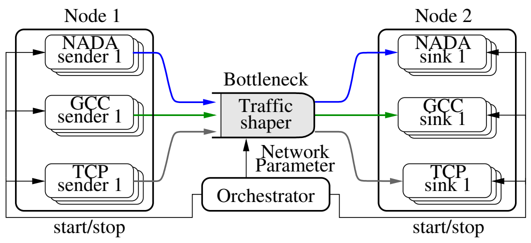

Fig. 5: Simulation Topology

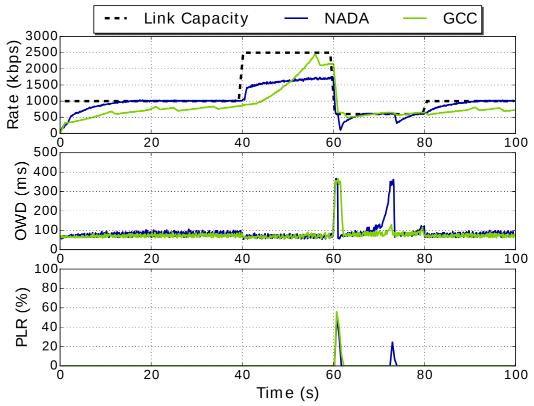

Fig. 6: Sending Rate, one-way delay (OWD) and packet loss ratio (PLR) dynamics in the case of a single GCC or NADA flow with variable link capacity

propagation delay on the direct path; for every simulation we compute the average value, the standard deviation, the 5th, 50th and 95th percentile; 4) Jain's Fairness Index: $J_{FI}(t) = \frac{(\sum_{k=1}^{n} r_k(t))^2}{n\sum_{k=1}^{n} r_k(t)^2}$, where $r_k(t)$ is the instantaneous throughput of the k-th flow and n is the total number of competing flows.

IV. Results

This section presents the results obtained employing the scenario described in Section III. The analysis is based on the evaluation criteria under discussion in the IETF WG RMCAT [19]. The goal is to check if both NADA and GCC satisfy the requirements defined in [18] i.e. low queuing in the absence of competing heterogeneous traffic and a reasonable share of bandwidth when competing with other homogeneous or heterogeneous flows. For every scenario ten simulations have been run. For each of them a Table is provided which contains the average value of the metrics measured in the ten runs.

A. *Single flow with variable link capacity*

This scenario investigates the dynamics of the video bitrate controlled by GCC or NADA to step-like variation of the link capacity. To the purpose, we set the link capacity to 1Mbps for the fist 40s, then we increase the capacity to 2.5Mbps for 20s, after we decrease the capacity to 0.6Mbps for 20s and finally we set the capacity to the initial value 1Mbps for 20s, such that the simulation lasts 100s. Table I shows the average value of the metrics measured throughout the ten simulations. Due to space constraints Figure 6 shows the overlapping dynamics of the GCC and NADA throughput, one-way delay and packet loss ratio. This simulation shows that when the link capacity increases to 2.5Mbps, GCC takes about 15 seconds to reach the new value of the link capacity whereas NADA throughput does not reach 2.5Mbps. On the other hand, at lower values of the link capacity NADA better utilizes the channel, even though, when the link capacity is set to 0.6Mpbs the instantaneous one-way delay exhibits a spike. Overall, both NADA and GCC are able to contain the queuing delay since the one-way delay dynamics is kept close to the one-way propagation delay which is equal to 50ms, except in correspondence to the link capacity drop. In this case the increment of the one-way delay and loss ratio forces the algorithms to reduce their sending bitrate in order to keep the queuing delay low. Overall we have measured an average value of the 50th percentile of the queuing delay equal to 26.6ms for GCC whereas 32.2ms for NADA.

	Channel Util.	**Queuing [ms]**			**Loss Ratio**
		5th	**50th**	**95th**	
GCC	76%	11	26.6	40	0.5%
NADA	79%	11	32.2	53	0.6%

TABLE I: Average value of the measured metrics in the case of a single GCC or NADA flow with variable link capacity

B. *Intra-protocol fairness*

The aim of this scenario is to investigate the GCC and NADA intra-protocol fairness. To the purpose, we have considered three concurrent GCC or NADA flows over a 3.5Mbps link. Each flow is started 20 seconds after the previous one. The simulation lasts 120 seconds. Table II shows the average value of the metrics measured over ten simulations when all the three flows are active. Figure 7 shows the dynamics of one simulation for NADA and GCC. We can notice that both NADA and GCC are not affected by the *"late-comer effect"* and they nicely share the bandwidth among the flows. Figure 7 shows that NADA convergence rate is faster than the GCC. GCC convergence rate can be intuitively shown by considering that the equivalent queuing delay variation measured by each flow is given by:

$$m(t) = m_{GCC1}(t) + m_{GCC2}(t) + m_{GCC3}(t) \qquad (8)$$

so that an increase of the queuing delay gradient induced by one of the flow triggers an "*overuse signal*" for all the concurrent flows. Since the bitrate is decreased according to the state machine shown in Figure 4, a flow with higher bitrate will experience a higher rate decrease, which leads to rate convergence. In the case of NADA all flows experience approximately the same aggregate congestion signal x_n (see Section II-A) at steady state. According to (4), the target rate at steady state R_n for each flow satisfies the following equation:

$$R_n = \frac{x_{ref} \cdot R_{max}}{x_n} \qquad (9)$$

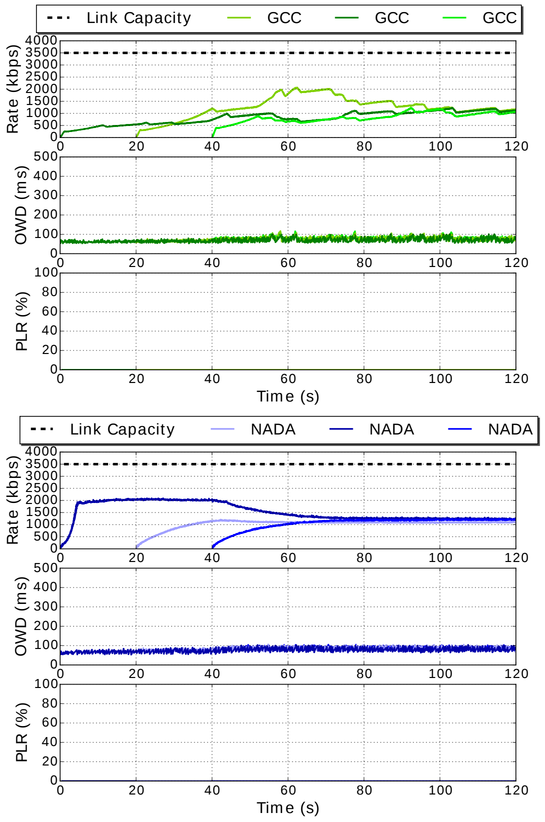

Fig. 7: Sending Rate, one-way delay (OWD) and packet loss ratio (PLR) dynamics in the case of three concurrent GCC or NADA flows over a 3.5Mbps link

Moreover, it is worth to notice that for both NADA and GCC no packets are lost during the whole duration of the simulation. Concerning the queuing delay, Table II shows an average value of the 50th percentile roughly equal to 20ms for both NADA and GCC flows. Overall the algorithms provide intra-protocol fairness: NADA shows a slightly better Jain Fairness index then GCC which are respectively equal to 0.95 and 0.87. To complete the investigation on the intra-protocol fairness we consider a scenario with 2 flows in the case of a link with variable capacity. The link capacity is varied every interval of 25 seconds. We consider 5 intervals in which the capacity is set respectively to 4Mbps, 2Mbps, 4Mbps, 1Mbps and 2Mbps, such that the video call lasts 125 seconds. Figure 8 shows the dynamics of the variables in one of the ten simulation runs carried out. In both of the cases Figure 8 shows that, when the link capacity changes, intra-protocol fairness is still maintained after a transient time. However, in this scenario the NADA sending rate exhibits remarkable oscillations when the link capacity drops to 1Mbps which also reflects on the one-way delay. Table III summarizes the average value of the metrics measured over the ten simulation; as expected the two GCC flows present a lower channel utilization but also lower queuing delay and loss ratio. The Jain Fairness index is higher than 0.84 for both of the algorithms confirming that they provide intra-protocol fairness.

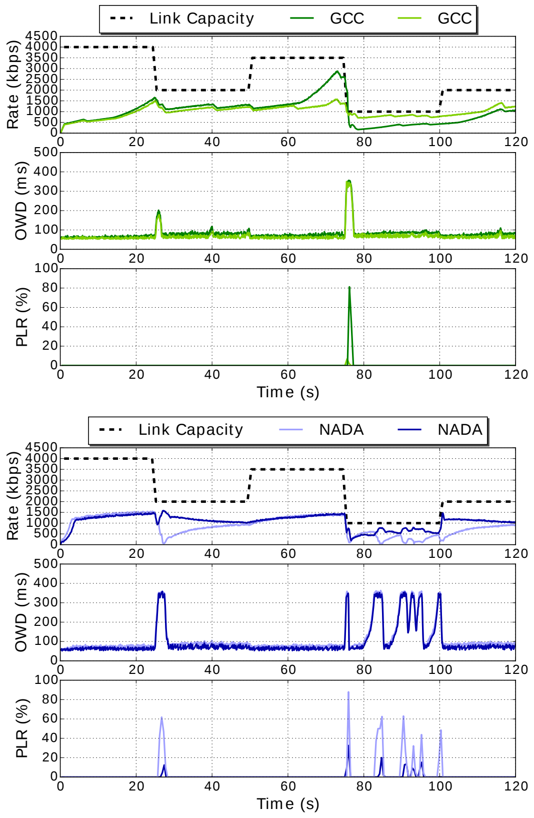

Fig. 8: Sending Rate, one-way delay (OWD) and packet loss ratio (PLR) dynamics in the case of two GCC or two NADA flows over a link with variable capacity

	Ch. Util.	Queuing [ms]			Loss Ratio	J_{FI}
		5th	50th	95th		
GCC	81%	6	20	48	0%	0.87
NADA	87%	7	20	50	0%	0.95

TABLE II: Average value of the measured metrics in in the case of three concurrent GCC or NADA flows over a 3.5Mbps link

C. *Inter-protocol Fairness*

This scenario aims at testing inter-protocol fairness requirements. To the purpose, we have considered one GCC or NADA flow against one TCP flow over a bottleneck link with a constant capacity equal to 2Mbps. The NADA or the GCC flow is started 5 seconds after the TCP flow. The simulation lasts 120 seconds. Table IV summarizes the metrics obtained in the ten simulations. Figure 9 shows the dynamics obtained by one of the simulation. In the case of GCC with a concurrent TCP flow the sending rate of the flows converges in about 50s to the fair share equal

	Ch. Util.	Queuing [ms]			Loss Ratio	J_{FI}
		5th	50th	95th		
GCC	74%	5	20	44	0.3%	0.84
NADA	77%	6	38	150	1%	0.85

TABLE III: Average value of the measured metrics in the case of two GCC or two NADA flows over a link with variable capacity

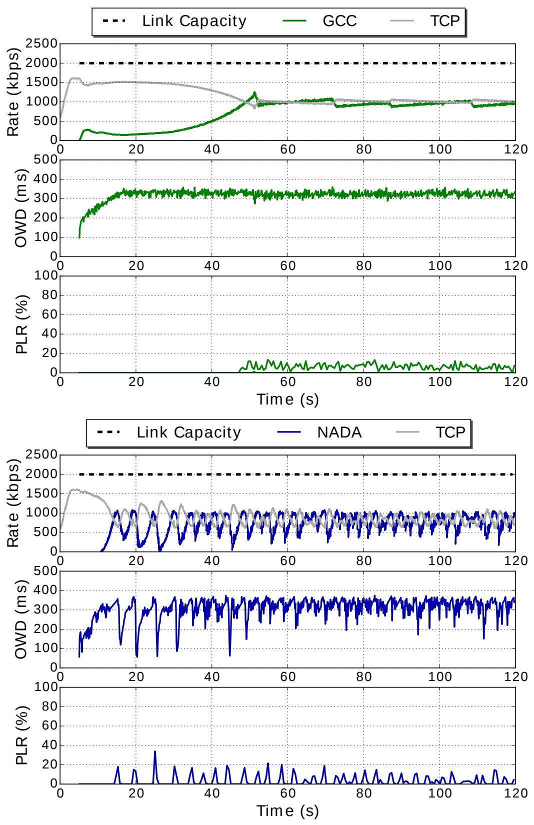

Fig. 9: The case of one GCC or NADA flow sharing a 2Mbps link with 1 TCP flow

	Tot. Ch. Util.	Loss Ratio	J_{FI}
GCC	98%	1.1%	0.80
NADA	97%	1.4%	0.81

TABLE IV: Average value of the measured metrics in the case of one GCC or NADA flow sharing a 2Mbps link with 1 TCP flow

to 1Mbps. This is possible due to the adaptive threshold design [5] that makes the delay-based controller less sensitive in the presence of concurrent loss-based traffic. In the case of NADA convergence is faster but NADA and TCP rate oscillate around the fair share at 1Mbps. As expected, the value of the fraction loss measured for both NADA and GCC is greater than zero in the presence of TCP since the algorithms have to operate in loss-based mode in oder to be more aggressive and compete against TCP. Finally, the queuing delay cannot be contained as expected, since the TCP traffic fills the bottleneck buffer.

V. Conclusion and Future Work

In this paper we have presented and compared the performance of NADA and GCC, two congestion control algorithms proposed for real-time communication over the Web. The simulations were carried based on the guidelines under definition in the IETF RMCAT working group. Results show that overall both the algorithms: 1) adapt the sending rate to track the link capacity, 2) provide intra-protocol and inter-protocol fairness. In particular NADA presents a faster convergence than GCC but exhibits oscillations when the link capacity drops below 0.6Mbps and in presence of concurrent TCP traffic. As future work we aim at finding the cause of NADA oscillatory behavior and improving GCC slow convergence.

References

[1] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communicationss*, 13(8):1465–1480, Oct. 1995.

[2] C. Briat, H. Hjalmarsson, K. H. Johansson, U. T. Jönsson, G. Karlsson, and H. Sandberg. Nonlinear state-dependent delay modeling and stability analysis of internet congestion control. In *Proc. of Conference on Decision and Control (CDC)*, pages 1484–1491. IEEE, 2010.

[3] C. Briat, E. A. Yavuz, H. Hjalmarsson, K. H. Johansson, U. T. Jonsson, G. Karlsson, and H. Sandberg. The conservation of information, towards an axiomatized modular modeling approach to congestion control. *IEEE/ACM Tran. on Networking*, 23(3):851–865, 2015.

[4] 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.

[5] G. Carlucci, L. De Cicco, and S. Mascolo. Modelling and Control for Web Real-Time Communication. In *53rd IEEE Conference on Decision and Control*, Los Angeles, CA, USA, Dec. 2014.

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

[7] 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.

[8] L. De Cicco and S. Mascolo. A mathematical model of the Skype VoIP congestion control algorithm. *IEEE Transactions on Automatic Control*, 55(3):790–795, Mar. 2010.

[9] L. De Cicco, S. Mascolo, and S.-I. Niculescu. Robust stability analysis of smith predictor-based congestion control algorithms for computer networks. *Automatica*, 47(8):1685–1692, 2011.

[10] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM CCR*, 36(1):59–62, Jan. 2006.

[11] L. A. Grieco and S. Mascolo. Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. *ACM SIGCOMM CCR*, 34(2):25–38, Apr. 2004.

[12] 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*, pages 328–341, Jul. 2011.

[13] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. A control theoretic analysis of red. In *Proc. of IEEE INFOCOM*, volume 3, pages 1510–1519 vol.3, Apr. 2001.

[14] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *IEEE Control Systems*, 22(1):28–43, 2002.

[15] S. Mascolo. Congestion control in high-speed communication networks using the smith principle. *Automatica*, 35(12):1921–1935, 1999.

[16] W. Michiels, D. Melchor-Aguilar, and S.-I. Niculescu. Stability analysis of some classes of tcp/aqm networks. *International Journal of Control*, 79(9):1136–1144, 2006.

[17] C. Parsa and J. Garcia-Luna-Aceves. Improving TCP congestion control over Internets with heterogeneous transmission media. In *Proc. of Network Protocols (ICNP '99)*, pages 213–221, Oct. 1999.

[18] J. Randell and Z. Sarker. Congestion control requirements for RMCAT. *Draft IETF*, Dec. 2014.

[19] Z. Sarker, V. Singh, X. Zhu, and R. M. Test Cases for Evaluating RMCAT Proposals. *IETF Draft*, Aug. 2015.

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

[21] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (LEDBAT). *RFC 6817*, Dec. 2012.

[22] R. Shorten, F. Wirth, and D. Leith. A positive systems model of tcp-like congestion control: asymptotic results. *IEEE/ACM Transactions on Networking*, 14(3):616–629, 2006.

[23] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006.

[24] X. Zhu, R. Pan, S. Mena, P. Jones, J. Fu, S. D'Aronco, and C. Ganzhorn. Nada: A unified congestion control scheme for real-time media. *IETF Draft*, Mar. 2015.