



Politecnico di Bari
Dipartimento di
Ingegneria Elettrica
e dell'Informazione



C3LAB
Control of Computing
and Communication
Systems Lab

Understanding the Dynamic Behaviour of the Google Congestion Control for RTCWeb

Packet Video Workshop - San Jose, CA, USA

12-13 December 2013

Luca De Cicco, Gaetano Carlucci, Saverio Mascolo



- ▶ Boosted by the large diffusion of mobile devices and high speed connections
- ▶ A bunch of real-time communication apps exist
- ▶ Each uses a proprietary set of algorithms
- ▶ Interoperability is not possible or not convenient



- ▶ Lots of apps converging on the web
- ▶ Why not leveraging web browsers for RTC?
- ▶ WebRTC (W3C) & RTCWeb (IETF) WGs are tackling this challenge



- ▶ Javascript API for HTML (W3C)
- ▶ Signalling & NAT traversal (IETF RTCWEB)
- ▶ Security (IETF RTCWEB)
- ▶ Congestion control (IETF RMCAT)



- ▶ Contain end-to-end delay (queuing delay)
- ▶ Contain packet losses (to decrease FEC)
- ▶ Reasonable fairness with other flows (both intra-protocol and inter-protocol)
- ▶ Prevent starvation when competing with TCP long-lived flows

See R. Jesup, “Congestion Control Requirements For RMCAT”, IETF Draft, Jul. 2013, [draft-ietf-rmcat-cc-requirements-00](#)



- ▶ WebRTC is now available in Chrome, Firefox, and Opera
- ▶ The only proposed congestion control algorithm for WebRTC that has been implemented in a browser is Google CC
- ▶ Reference implementation in Chrome (Chromium) stable

Experimentally investigate the dynamics and issues of the Google Congestion Control (GCC)

Lundin, Holmer, Alvestrand, "A Google Congestion Control Algorithm for Real-Time Communication", IETF RtcWeb -Rmcat, 2013



GOOGLE CONGESTION CONTROL

THE CONTROL ARCHITECTURE



- ▶ Audio/video flows sent using RTP over UDP
- ▶ Hybrid loss-based delay-based approach
- ▶ The sender-side controller probes the available bandwidth
- ▶ The receiver-side controller computes the "Receiver Estimated Maximum Bitrate" A_r to limit the sending rate A_s and contain the queuing delay

SENDING RATE COMPUTATION IN A NUTSHELL

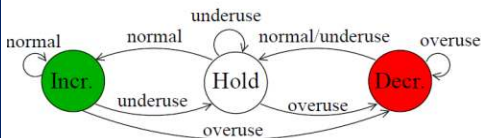
Sender-side: A_s

When a RTCP report is received:

- ▶ Low losses: multiplicative increase
- ▶ High losses: multiplicative decrease proportionally to f_l
- ▶ Moderate losses: constant

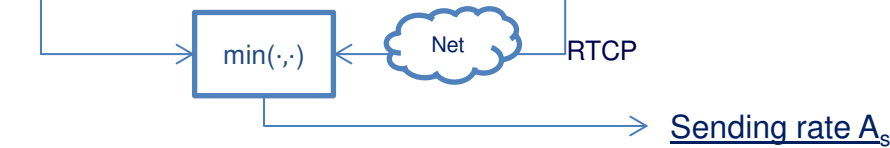
Receiver-side: A_r

Driven by a three states FSM



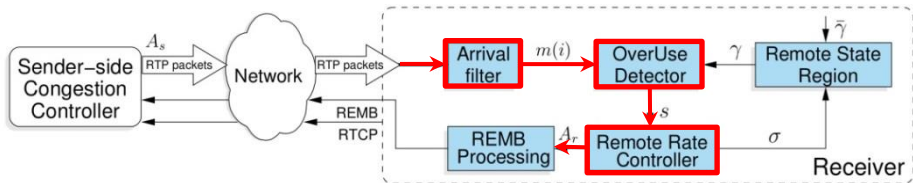
$$A_s(t_k) = \begin{cases} 1.05(A_s(t_{k-1}) + 1\text{kbps}) & f_l(t_k) < 0.02 \\ A_s(t_{k-1})(1 - 0.5f_l(t_k)) & f_l(t_k) > 0.1 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases}$$

$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \text{Increase} \\ \alpha R(t_i) & \text{Decrease} \\ A_r(t_{i-1}) & \text{Hold} \end{cases}$$



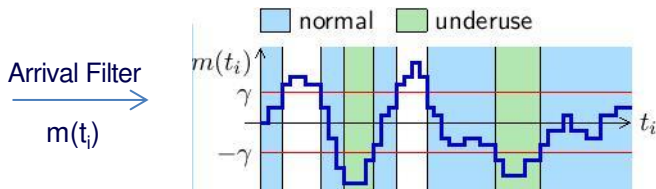
THE RECEIVER

IDEA: to compute the rate A_r based on the estimated one-way queuing-delay variation m



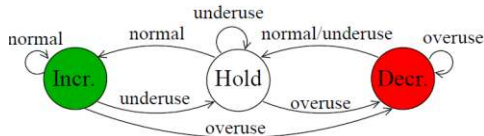
- ▶ Arrival filter: estimates the QD variation $m(t_i)$ using a Kalman filter
- ▶ Overuse detector (OUD): based on $m(t_i)$ generates a signal
- ▶ Remote rate controller: the signal produced by the OUD drives the state of a FSM that sets the rate A_r

THE OVERUSE DETECTOR



The signal is compared to two thresholds $-\gamma, \gamma$

- ▶ $|m| \leq \gamma$: the network is considered uncongested \Rightarrow "normal"
- ▶ $m < -\gamma$: the network is considered underused \Rightarrow "underuse"
- ▶ $m > \gamma$: for more then t_{OU} (100 ms) the network is considered congested \Rightarrow "overuse"

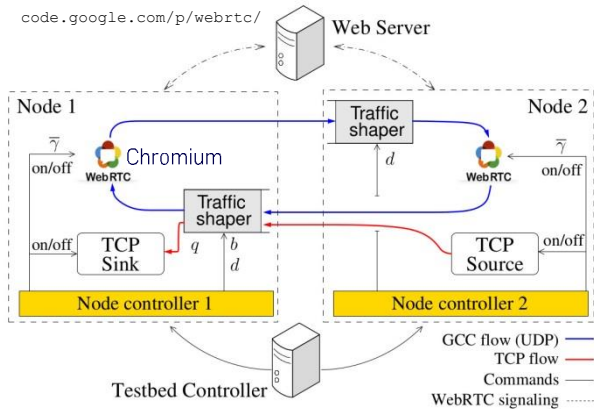


A overuse always triggers a switch to Decrease state



TESTBED

TESTBED



- ▶ Two hosts, one web server, one testbed controller to orchestrate experiments
- ▶ TC+NetEm to set bandwidth, buffer sizes, and base RTT
- ▶ TCP long-lived traffic using iperf
- ▶ Modified Chromium to change γ and log internal variables

METRICS

- ▶ Channel Utilization $U=R/b$: where b is the known available bandwidth and R is the average received rate
- ▶ Loss Ratio $l=(\text{packet lost})/(\text{packet received})$
- ▶ Queuing delay T_q : measured averaging the value $RTT(t)-RTT_m$ over all the RTT samples reported in the RTCP feedbacks
- ▶ Number of delay based decreased event n_{dd} : the number of times the remote controller makes the sending rate to be decreased.

TESTBED PARAMETERS

Single flow scenario

Bandwidth (kbps)	Buffer size (kB)	RTT (ms)	Threshold γ (ms)
500, 1000, 1500, 2000	15, 30, 45, 60, 75	50	10/60, 15/60, ..., 65/60

GCC vs TCP

Bandwidth (kbps)	Buffer size (kB)	RTT (ms)	Threshold γ (ms)
1000, 2000, 3000	15, 30, 45, 60, 75	50	10/60, 15/60, ..., 65/60

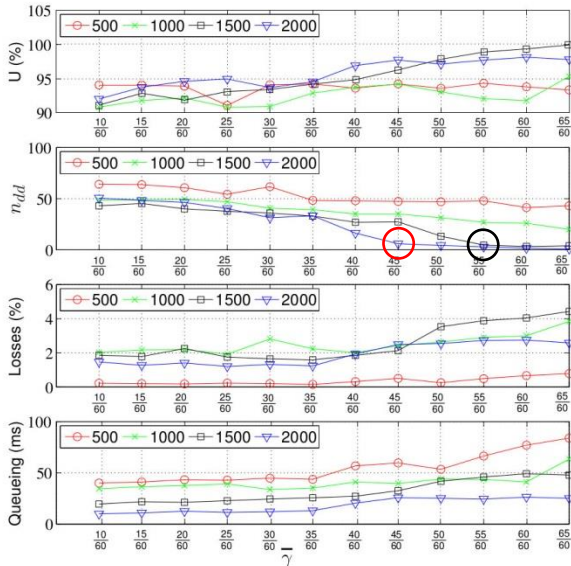
For each tuple of parameters we run three experiments

Dataset : 120 hours of active measurements, around 1300 calls



EXPERIMENTAL RESULTS

ONE GCC FLOW OVER A BOTTLECK (buffer size 30 kB)



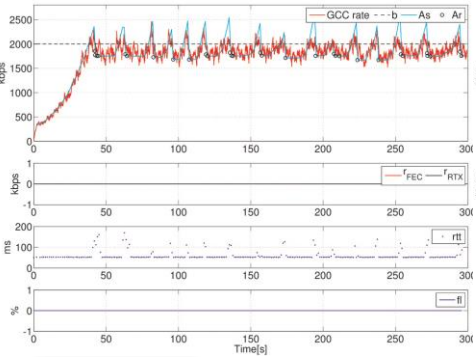
$\gamma=25/60$ default value

Utilization increases with γ

The number of delay-based decrease events decreases with γ

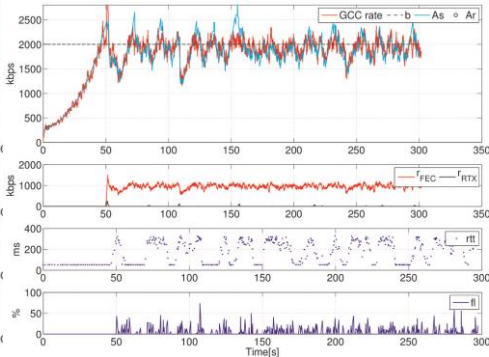
n_{dd} decreases faster at higher bandwidths

When n_{dd} decreases losses and queueing delay increase (more loss-based)

EFFECT OF γ ON QUEUING AND LOSSES (BW=2Mbps, QS=75kB)

$\gamma=10/60$

No losses, queuing delay less than 150 ms, several DB decrease events

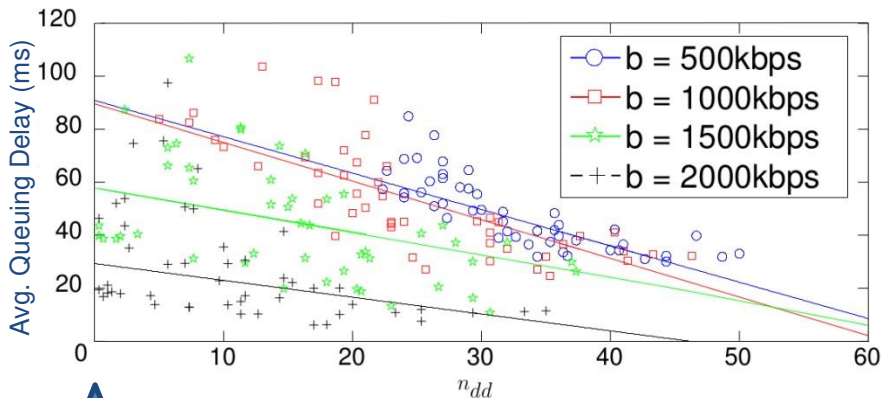


$\gamma=65/60$

Losses (4%), queuing delay doubled, zero DB decrease events. No influence of the DB algo

ONE GCC FLOW OVER A BOTTLECK

Bandwidth (kbps)	Base rtt (ms)	Runs	Buffer size kB	γ (ms)
500, 1000, 1500, 2000	50	3	15, 30, 45, 60, 75	[10/60-65/60]

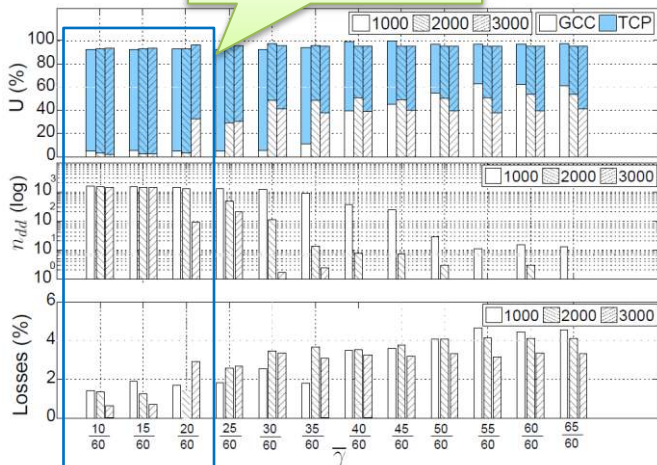


Loss-based only

Increasing influence of delay-based algo

ONE GCC FLOW vs ONE TCP FLOW (QS=30 kB)

Why starvation occurs?

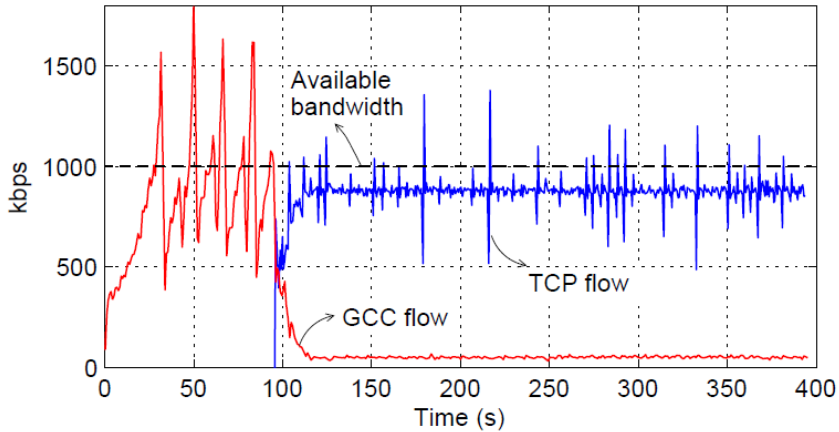


Large number of delay-based decrease events

With increased queue sizes a larger and larger γ is required to avoid starvation of GCC

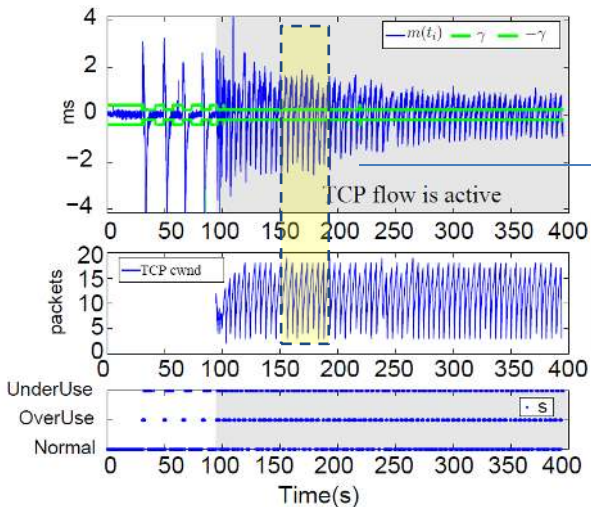
For low γ GCC is starved

GCC STARVED BY TCP (1/4)

 $b=1000\text{kbps}$, $q=30\text{kB}$, $\gamma=25/60$ 

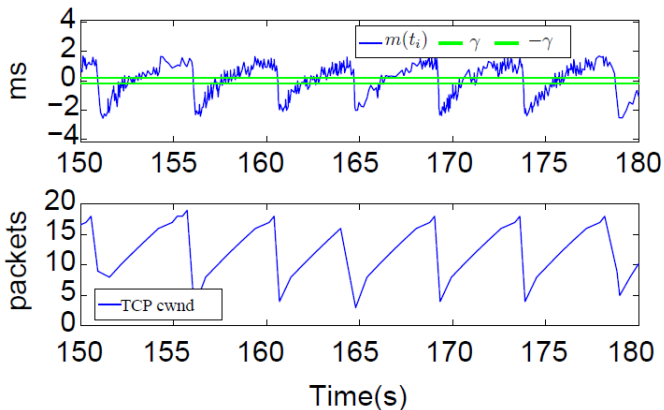
GCC STARVED BY TCP (2/4)

The receiver dynamics



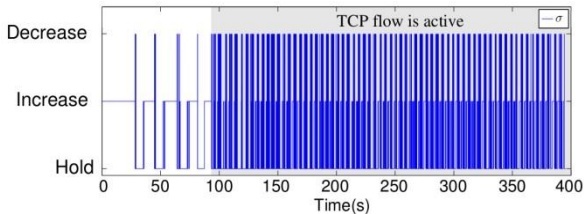
Large number of overuse signals produced after the TCP flow joins

GCC STARVED BY TCP (3/4)

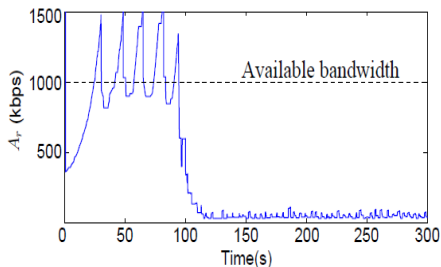


large QD variations due to TCP cong. control dynamics and not to self-inflicted delay

Remote rate controller state



When TCP is on the Remote Rate controller FSM switches to Decrease mode frequently



A_r is quickly decreased, leading to starvation

CONCLUSIONS

Single GCC flow

- ▶ the threshold γ has a remarkable impact on the performance
- ▶ for lower value of γ queuing and losses are contained

GCC vs TCP

- ▶ the threshold γ has a remarkable impact on the friendliness
- ▶ for sufficiently high value of γ reasonable fairness is reached

The threshold should be made adaptive to provide optimal performance and prevent starvation in the case of concurrent TCP traffic



QUESTIONS?

BACKUP SLIDES

One way delay variation model

OWD var. = Transm. Time var. + Queuing delay var. + net jitter

$$d(t_i) = \frac{L(t_i) - L(t_{i-1})}{C(t_i)} + m(t_i) + n(t_i)$$

$d(t_i)$ ← i-th frame size
 $L(t_i) - L(t_{i-1})$
 $C(t_i)$ ← Bottleneck bw
 $m(t_i)$
 $n(t_i)$ ← considered gaussian

One way delay variation measurement

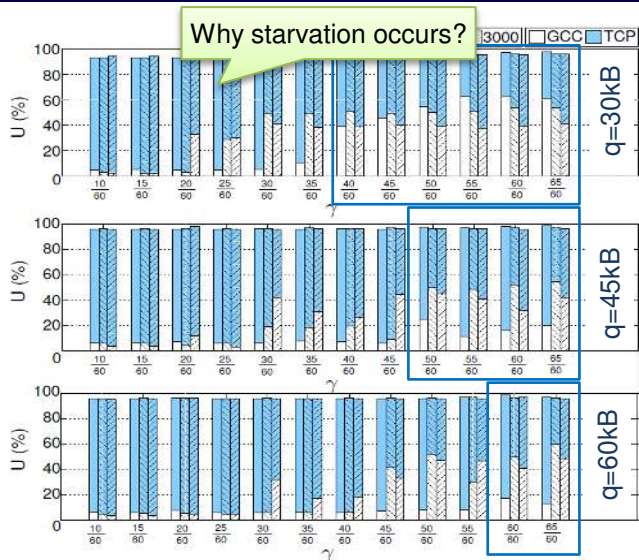
OWD var. = Inter-arrival time - Inter-departure time

$$d_m(t_i) = t_i - t_{i-1} - T_i + T_{i-1}$$

QD variation $m(t_i)$ computation

A Kalman filter computes $m(t_i)$ and $1/C(t_i)$ to steer the residual measurement error $d(t_i) - d_m(t_i)$ to zero

CHANNEL UTILIZATION vs QUEUE SIZE



With increased queue sizes a larger and larger γ is required to avoid the starvation of GCC flows