Politecnico di Bari
Dipartimento di Ingegneria Elettrica e dell'Informazione

C3LAB
Control of Computing and Communication Systems Lab

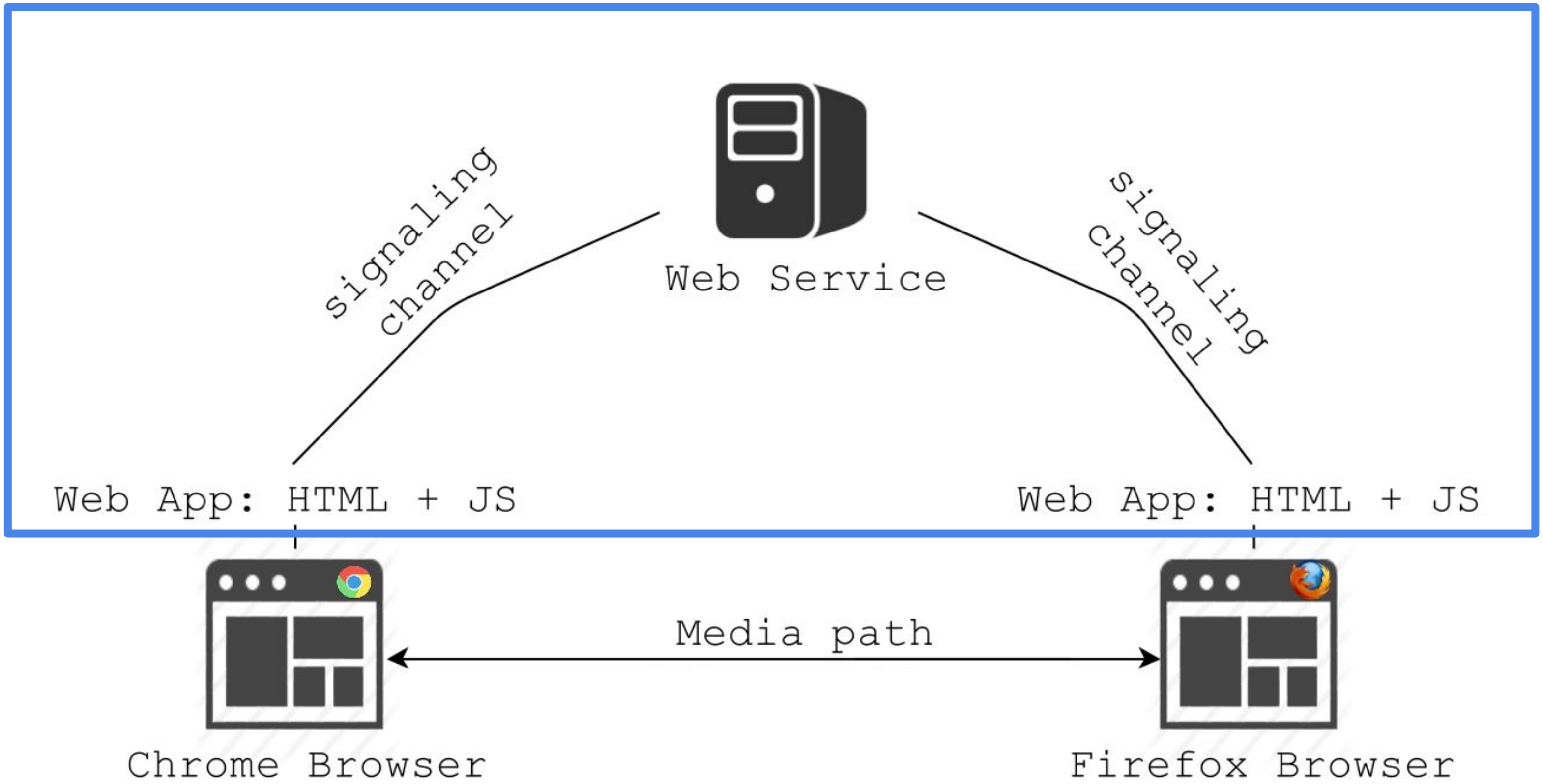
Congestion Control for WebRTC

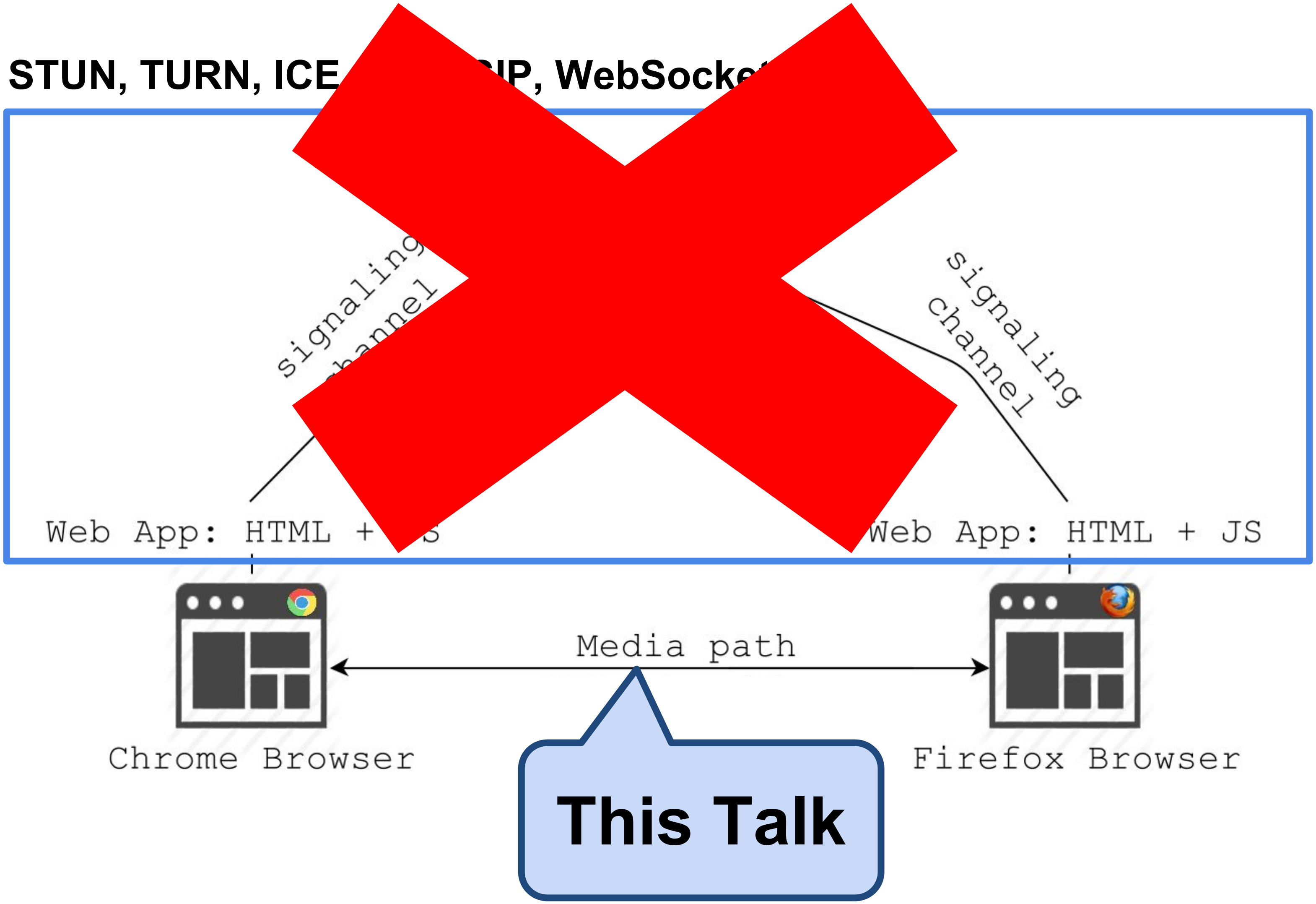
TF-WebRTC
22 September 2016

Luca De Cicco
luca.decicco@poliba.it

Politecnico di Bari, Italy

STUN, TURN, ICE, SDP, SIP, WebSockets ...

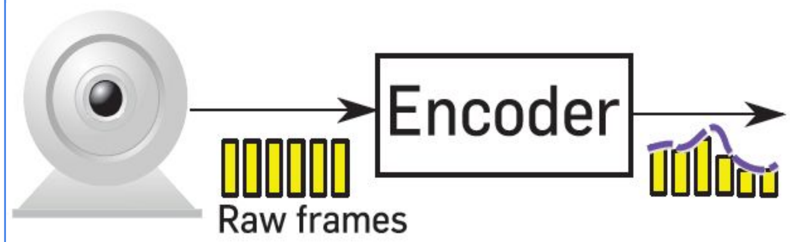
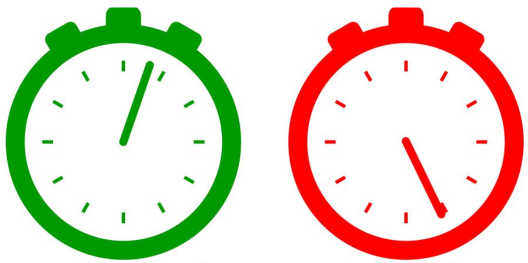



STUN, TURN, ICE
signaling
signaling channel
Web App: HTML + JS
Media path
Chrome Browser
Firefox Browser
This Talk

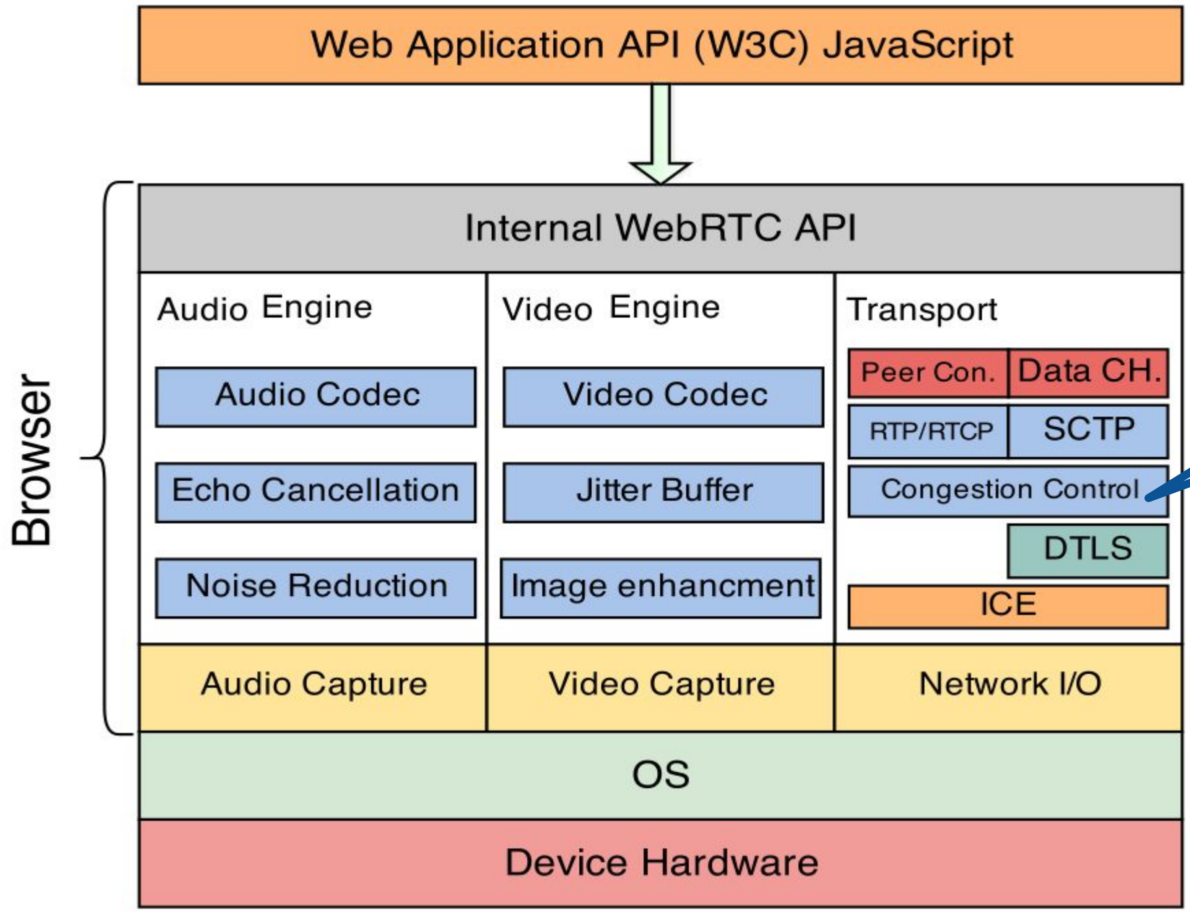
AGENDA

- Congestion Control for Real-Time media
- Google Congestion Control
- Tools to experiment with GCC
- Some results

- Long standing research area bootstrapped by Van Jacobson in the 80s
- Many approaches to do it depending on traffic to transport and the underlying network
- … it boils down to **regulating the sending rate of an application so that congestion is avoided**

Bulk data	**Media flows**
	
 Good Bad	 Good Bad
TCP for reliability	**UDP:** can trade some losses for timeliness

WEBRTC STACK

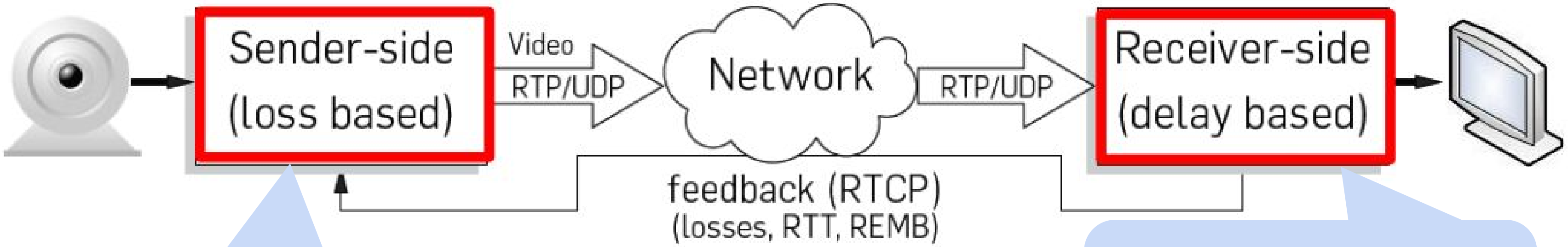


Focus of this talk

Congestion Control for audio/video

Note: the dataChannel is not affected

GOOGLE CONGESTION CONTROL ARCHITECTURE



Fall-back

Most part of the job done here*

- Audio/video flows sent using RTP/UDP (feedback over RTCP)
- The receiver employs a **delay-based** controller with the aim of containing queuing delays
- The sender employs **loss-based** controller
- Used in and

* currently

CONGESTION METRIC

Control goal

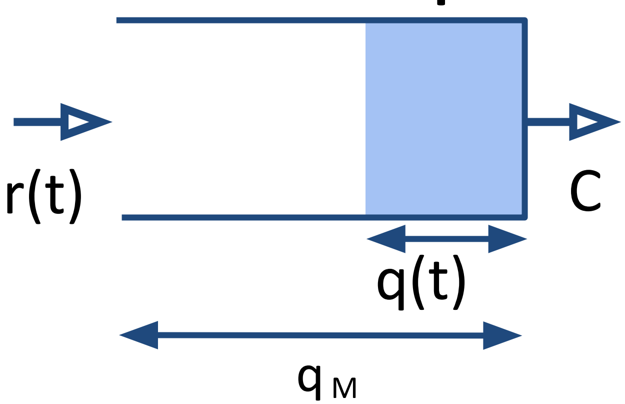
Avoiding packet losses while **controlling delays**

One Way Delay = One Way Propagation Delay + **Queuing Delay**

Cannot do anything E2E

Can control this

Bottleneck link queue



$$\text{Queuing delay} = \frac{\text{Inst. queue level}}{\text{Capacity}}$$

1. Estimate the OWDV
2. Congestion Detection
3. Compute the rate

ESTIMATION

- **One Way Delay** is affected by offsets (*late-comer*)
- We propose to use the **OWD variation** (OWDV)

$$OWDV_k = OWD_k - OWD_{k-1}$$

- Intuitively:

$OWDV_k > 0 \Rightarrow$ delay $\uparrow$ rate should be **decreased**

$OWDV_k < 0 \Rightarrow$ delay $\downarrow$ rate should be **increased**

What about $OWDV_k = 0$?

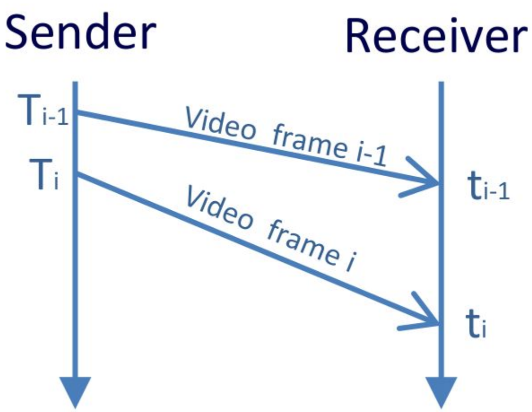
OWDV = 0

What happens when **OWDV = 0** at steady-state (i.e. $OWD_k = OWD_{k\text{-}1}$ for all k)?

- **persistent congestion** (queue full)
- **persistent underutilization** (queue empty)
- **persistent standing queue** (queue constant)

Some (delay-based) probing is needed

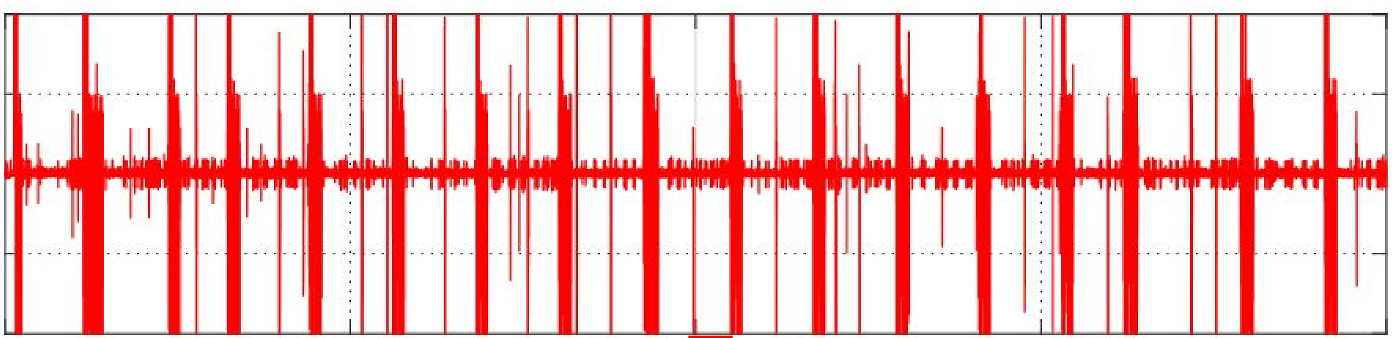
OWDV ESTIMATION



$$OWDV_i = OWD_i - OWD_{i-1} =$$
$$= (t_i - T_i) - (t_{i-1} - T_{i-1}) =$$
$$= (t_i - t_{i-1}) - (T_i - T_{i-1})$$

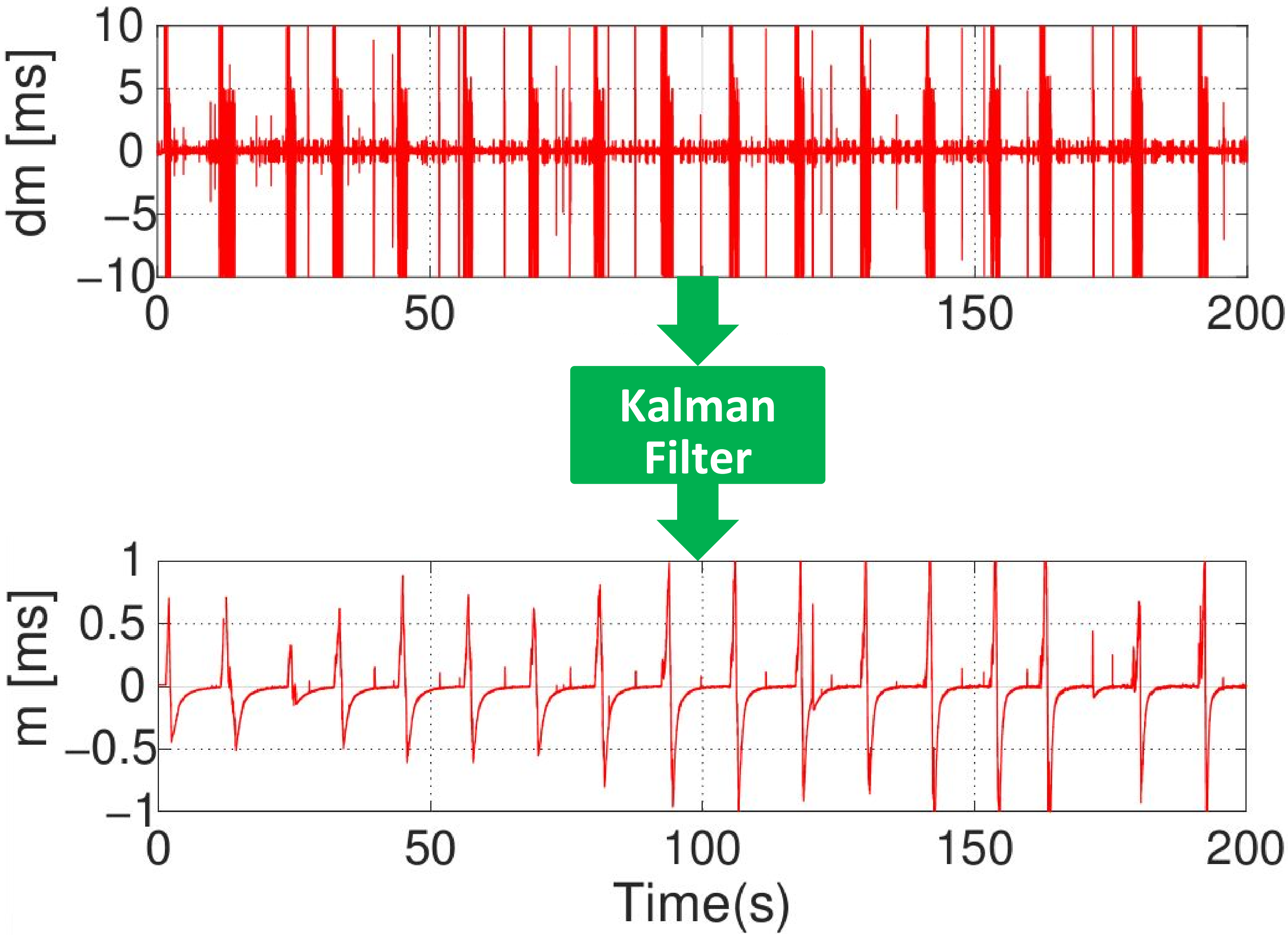
Inter-arrival time

Inter-depar. time



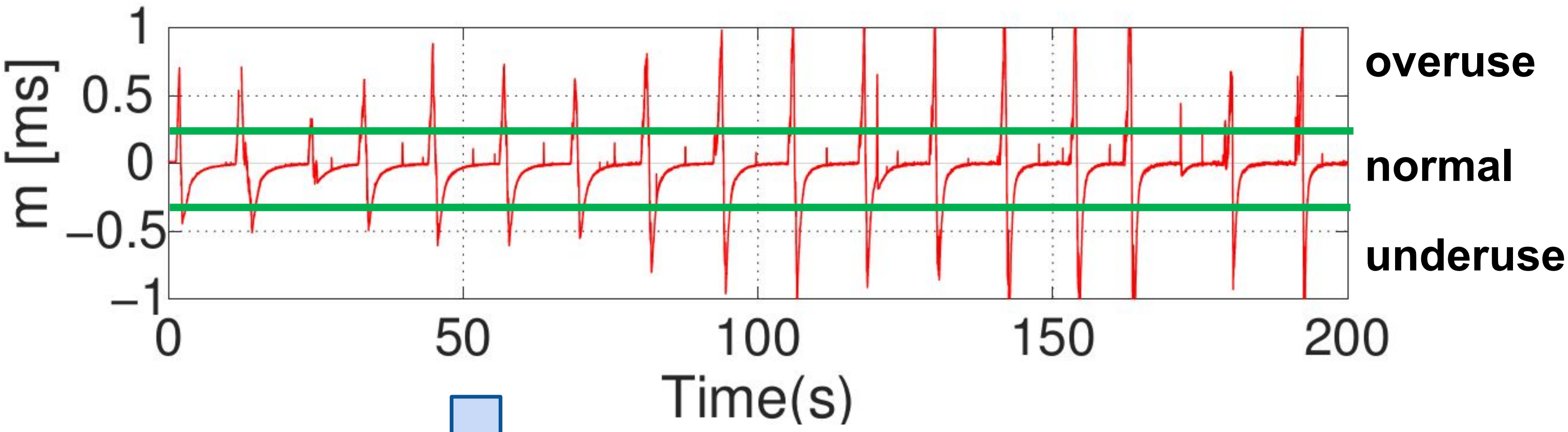
The measured OWDV is noisy

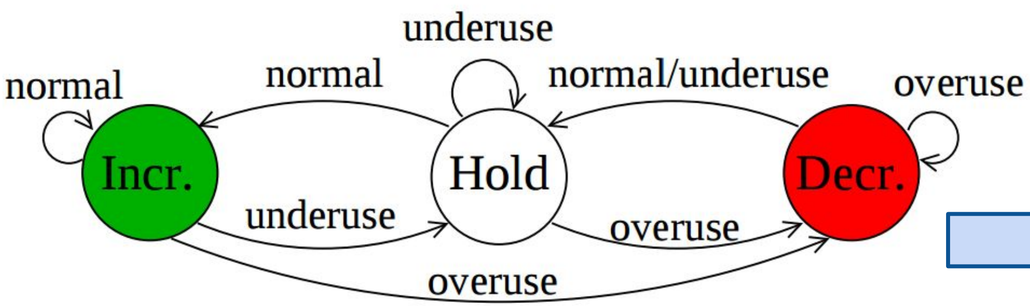
OWDV ESTIMATION



CONGESTION DETECTION (FIST ATTEMPT)

The estimated OWDV is compared to a threshold to detect congestion





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

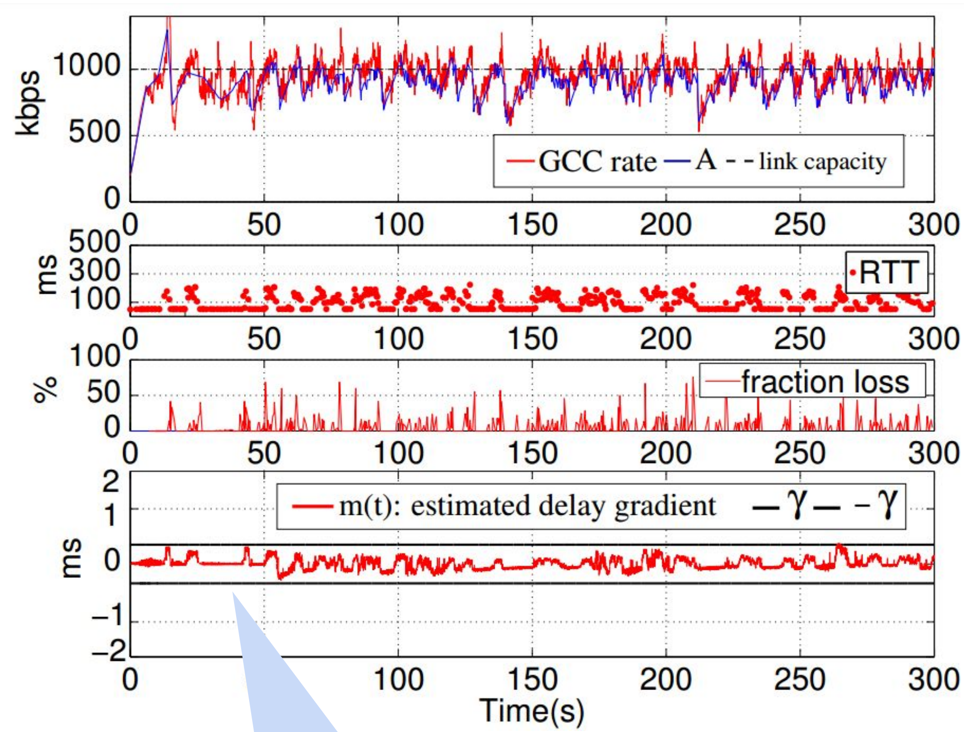
- IETF RMCAT draft: draft-alvestrand-rmcat-remb-03
- Specifies the RTCP feedback message to signal the rate computed by GCC delay-based algo

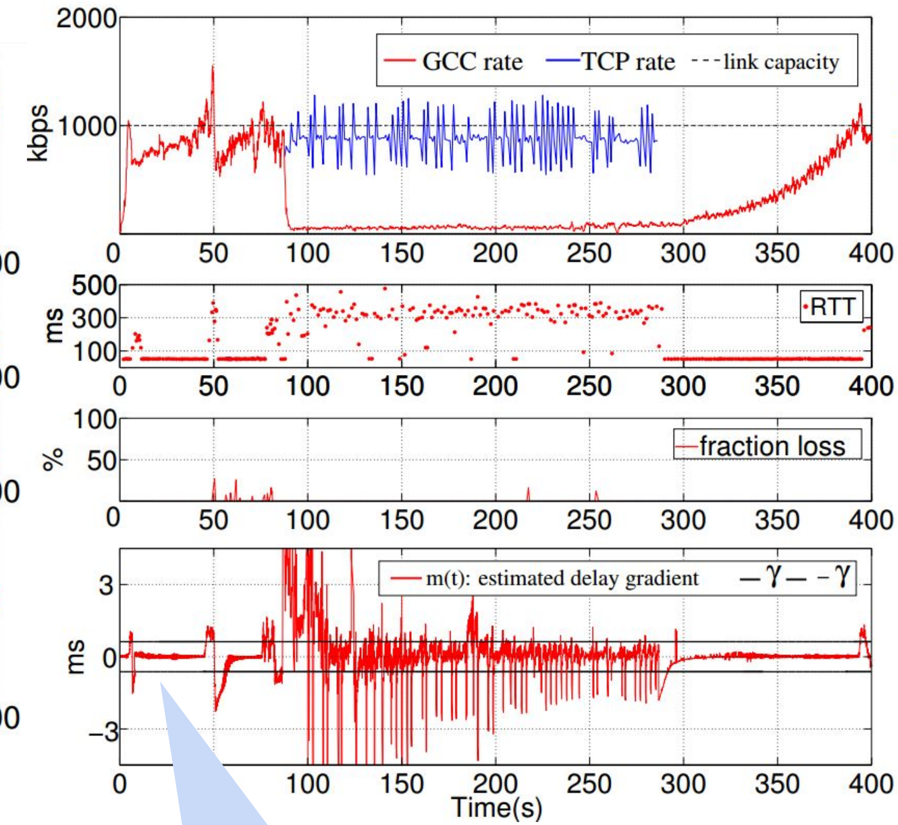
> *"The reception of a REMB message [...] SHALL result in the total bit rate sent on the RTP session this message applies to being **equal to or lower than the bit rate in this message**. [...] The **sender** is free to apply **additional bandwidth restrictions** based on its own restrictions and estimates."*

- In other words: **the sending rate will not exceed the rate stamped in REMB messages**

CONGESTION DETECTION WITH A STATIC THRESHOLD: BAD NEWS

There is no static *"right"* value for the threshold





Too large!

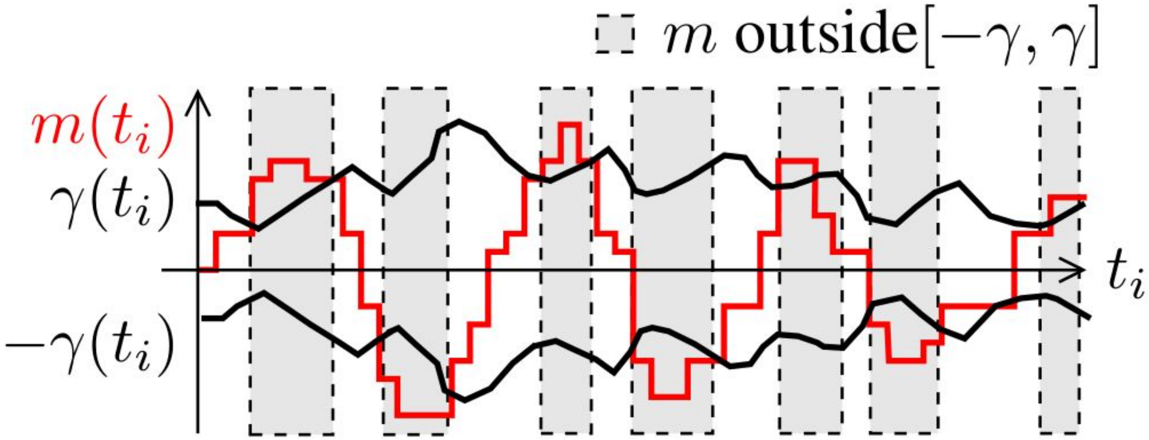
Too small!

SOLUTION: MAKE THE THRESHOLD ADAPTIVE

Idea

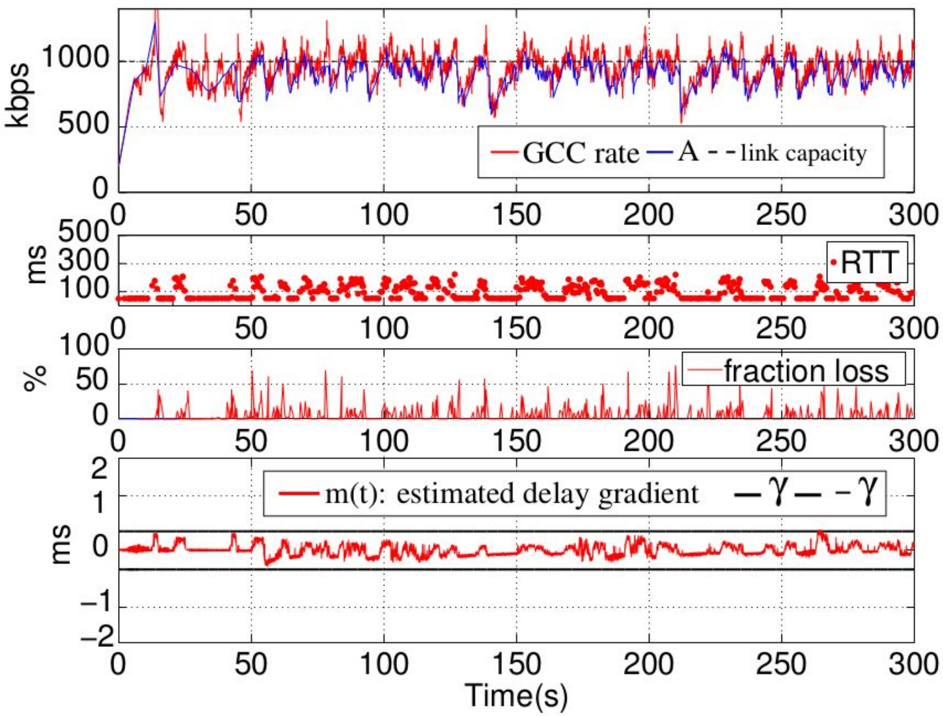
- Increase the threshold when *m* is outside the range [-γ,γ]
- Decrease the threshold when *m* falls back into [-γ,γ]

$$\frac{d\gamma(t)}{dt} = k(t)(|m(t)| - \gamma(t))$$

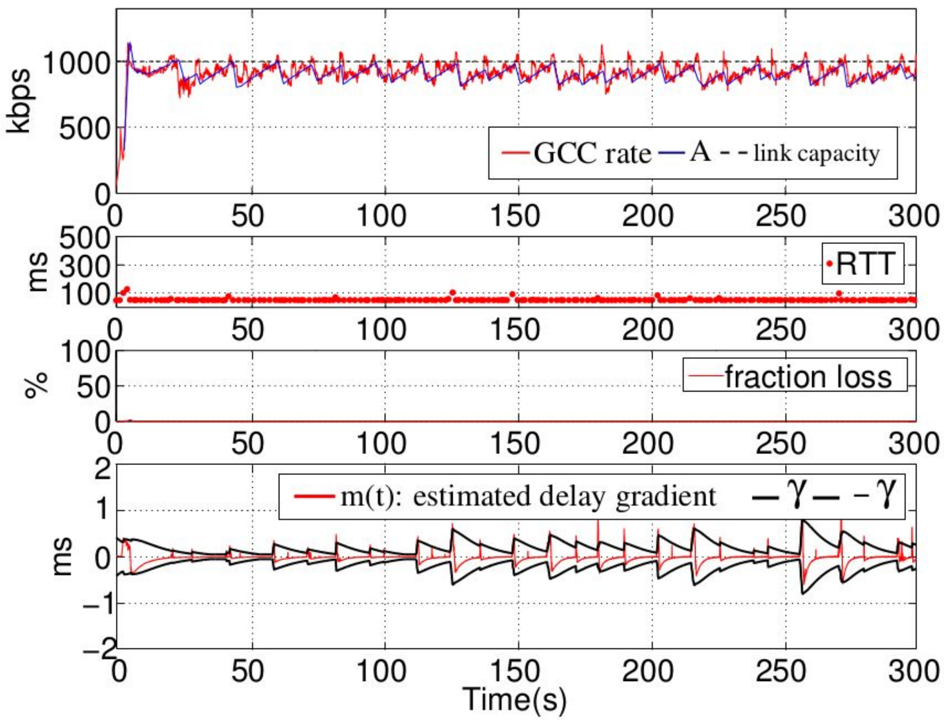


ADAPTIVE THRESHOLD RESULTS: GOOD NEWS

STATIC



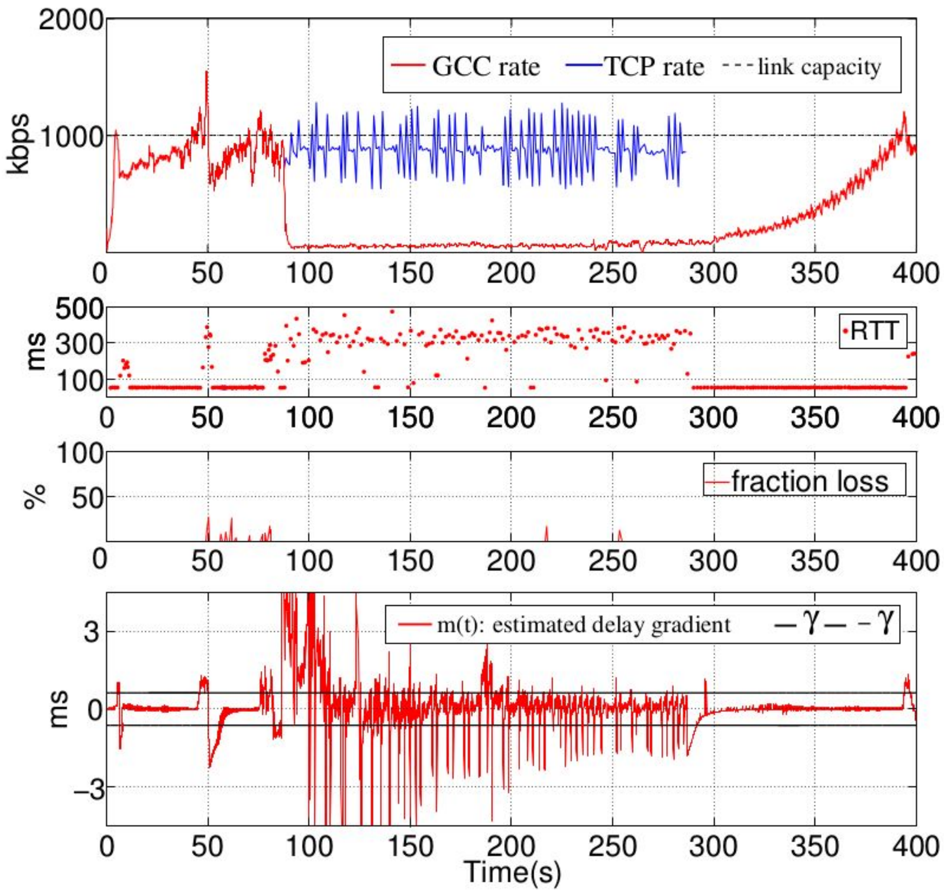
ADAPTIVE



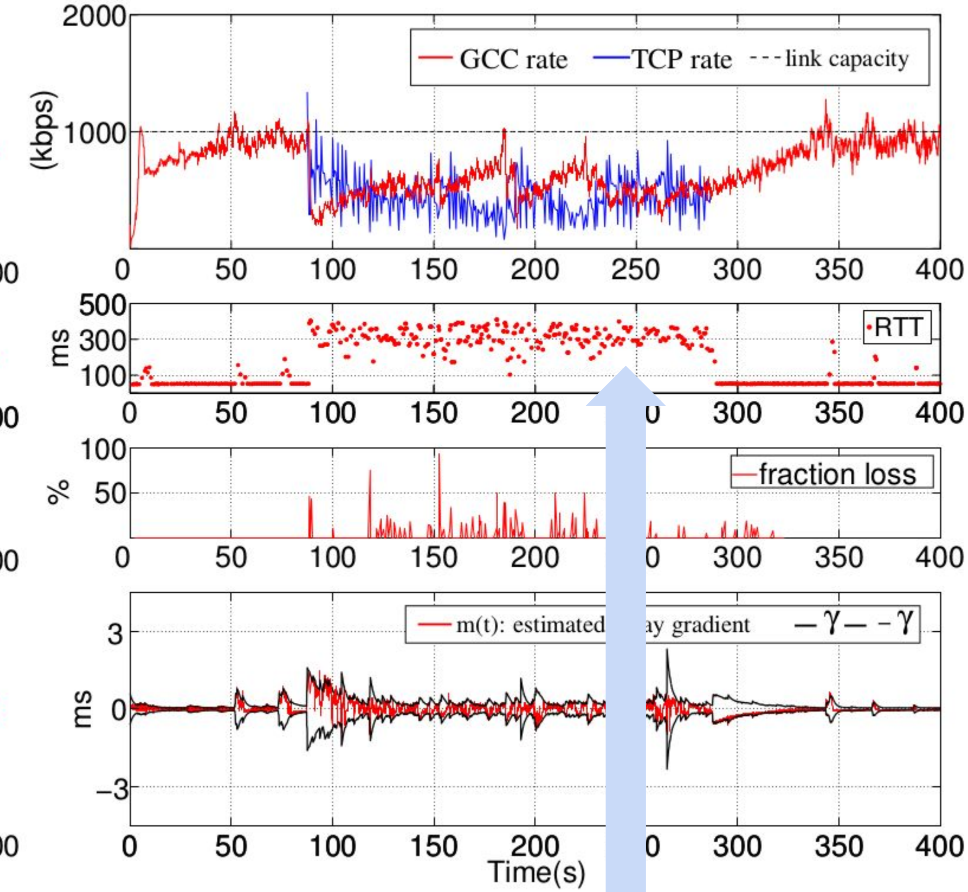
The adaptive law finds the *"right"* value for the threshold

ADAPTIVE THRESHOLD RESULTS

STATIC

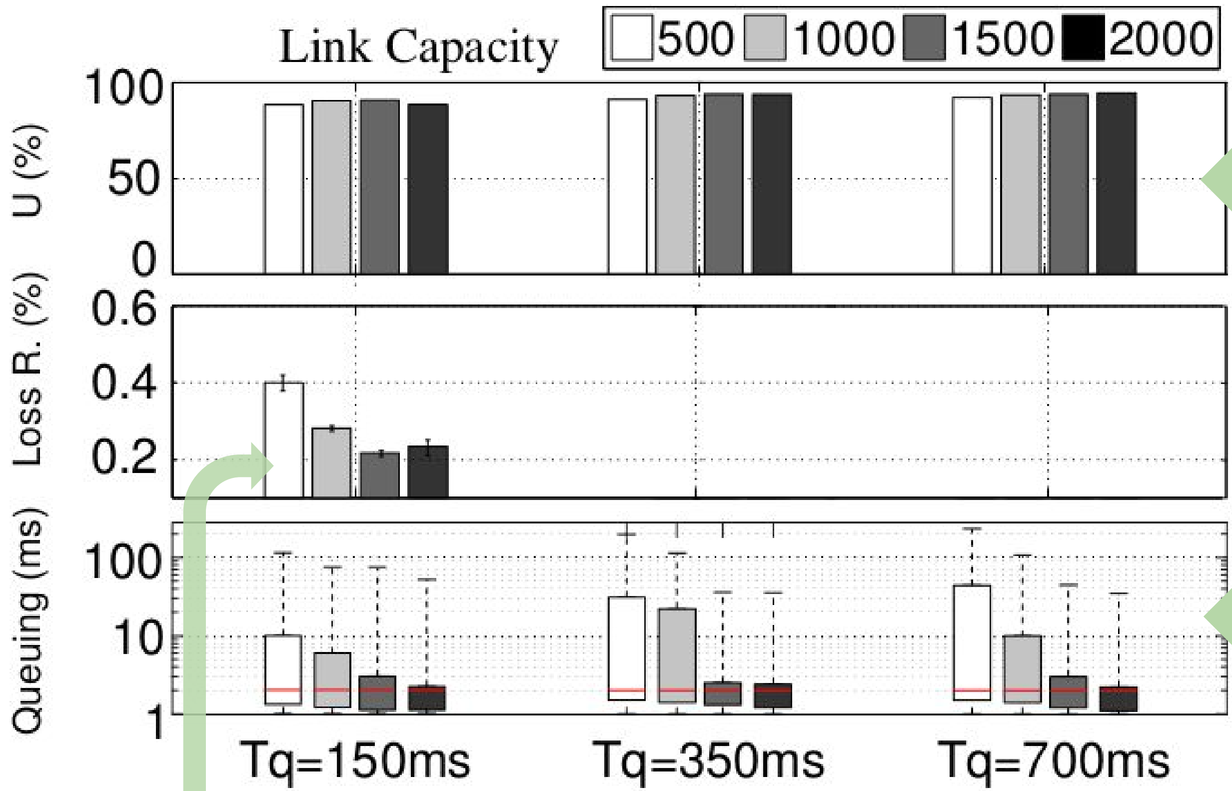


ADAPTIVE



Can compete with a TCP greedy flow (but can't perform miracles)

SINGLE FLOW - CONSTANT CAPACITY - DIFFERENT QUEUE SIZES



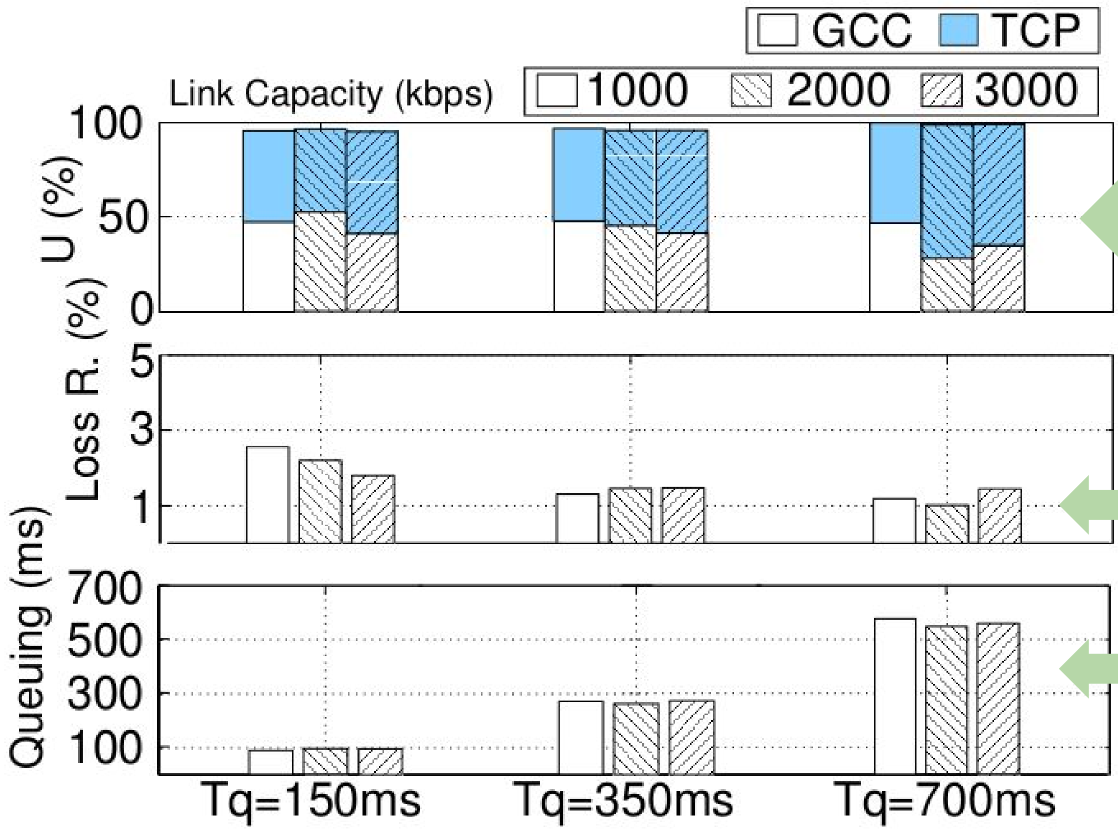
High link utilization

Low queuing

Log scale

(Negligible) losses only when the queue is small (due to the "slow-start" phase)

GCC VS TCP - CONSTANT CAPACITY - DIFFERENT QUEUE SIZES



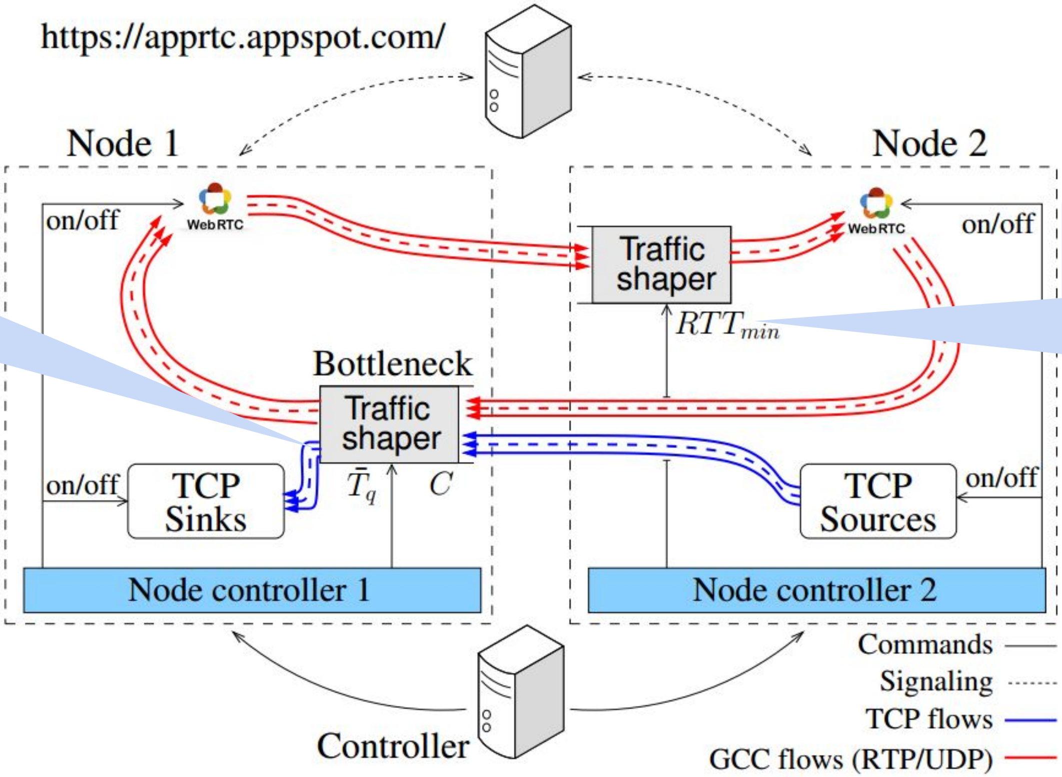
Fair sharing

Cannot perform miracles: losses and queuing due to competing TCP flow

EXPERIMENTING WITH GCC: TESTBED

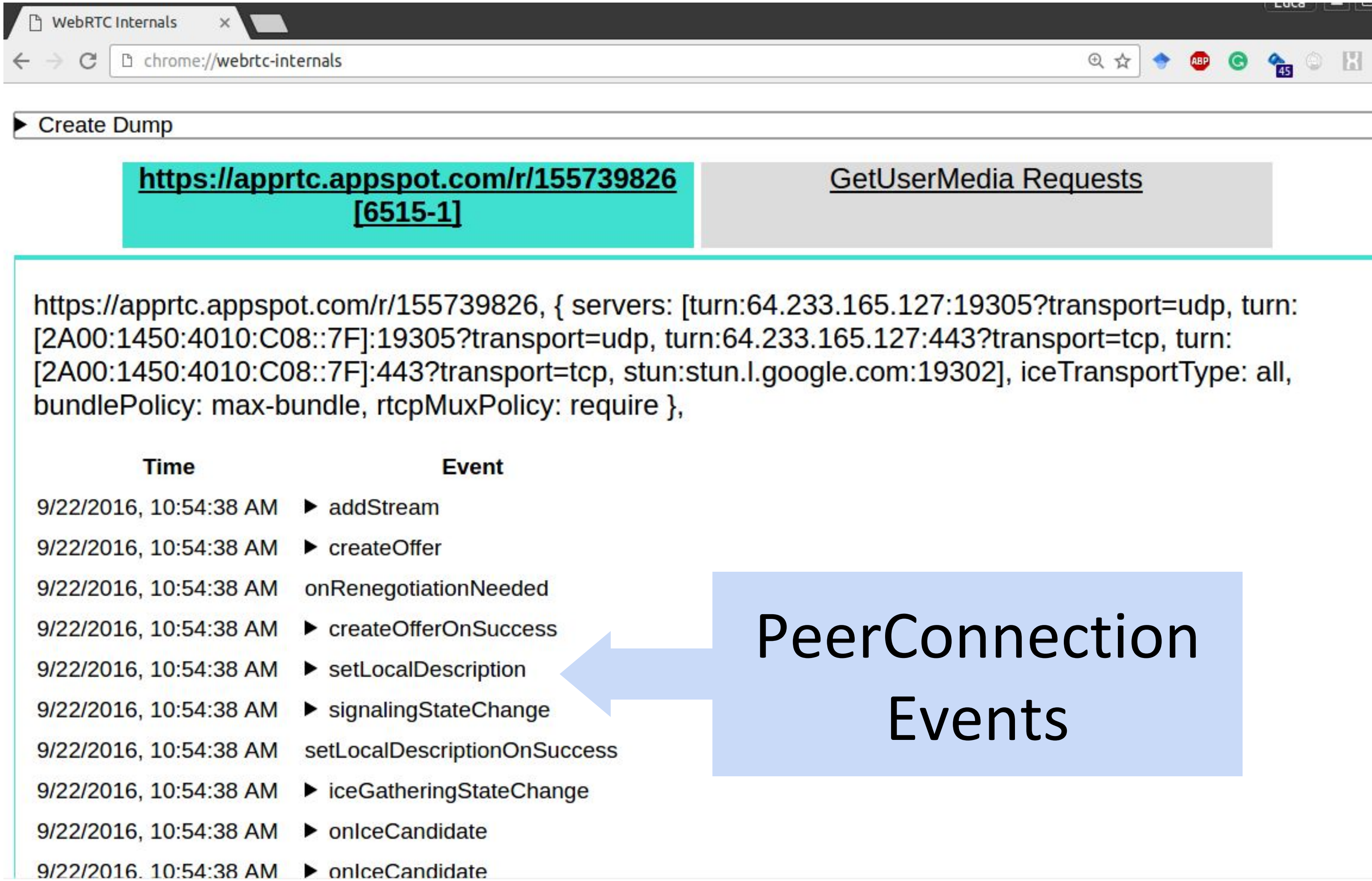
- Video Encoder: VP8
- Video sequence: https://people.xiph.org/~thdavies/x264_streams/
- Signalling: https://apprtc.appspot.com/ (modified)
- Wan Emulation Helpers: GitHub repository

`tc` + TBF
to set T_q and C

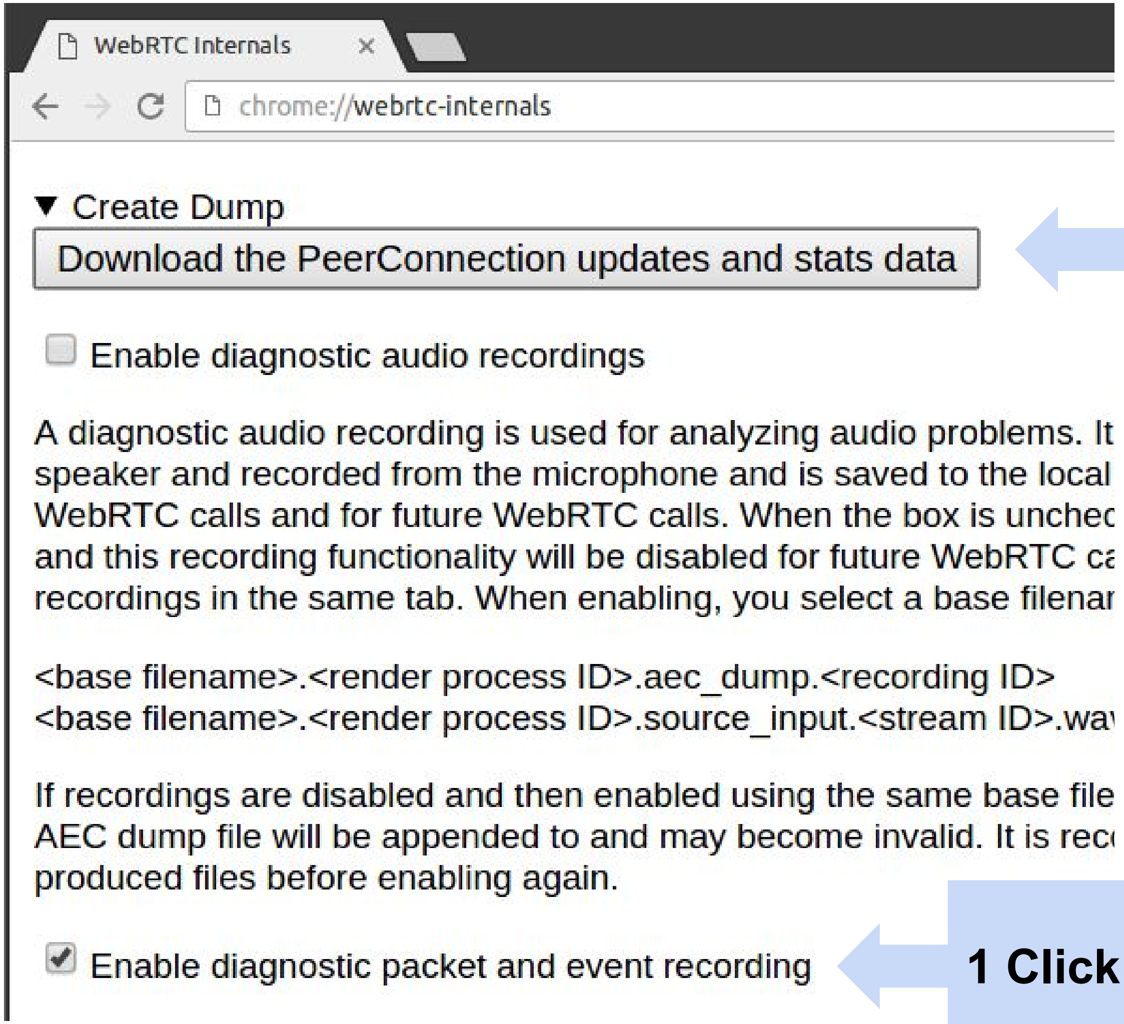


`NetEm`
to set
prop. delay

chrome://webrtc-internals



chrome://webrtc-internals: Dump diagnostic



3. Click here

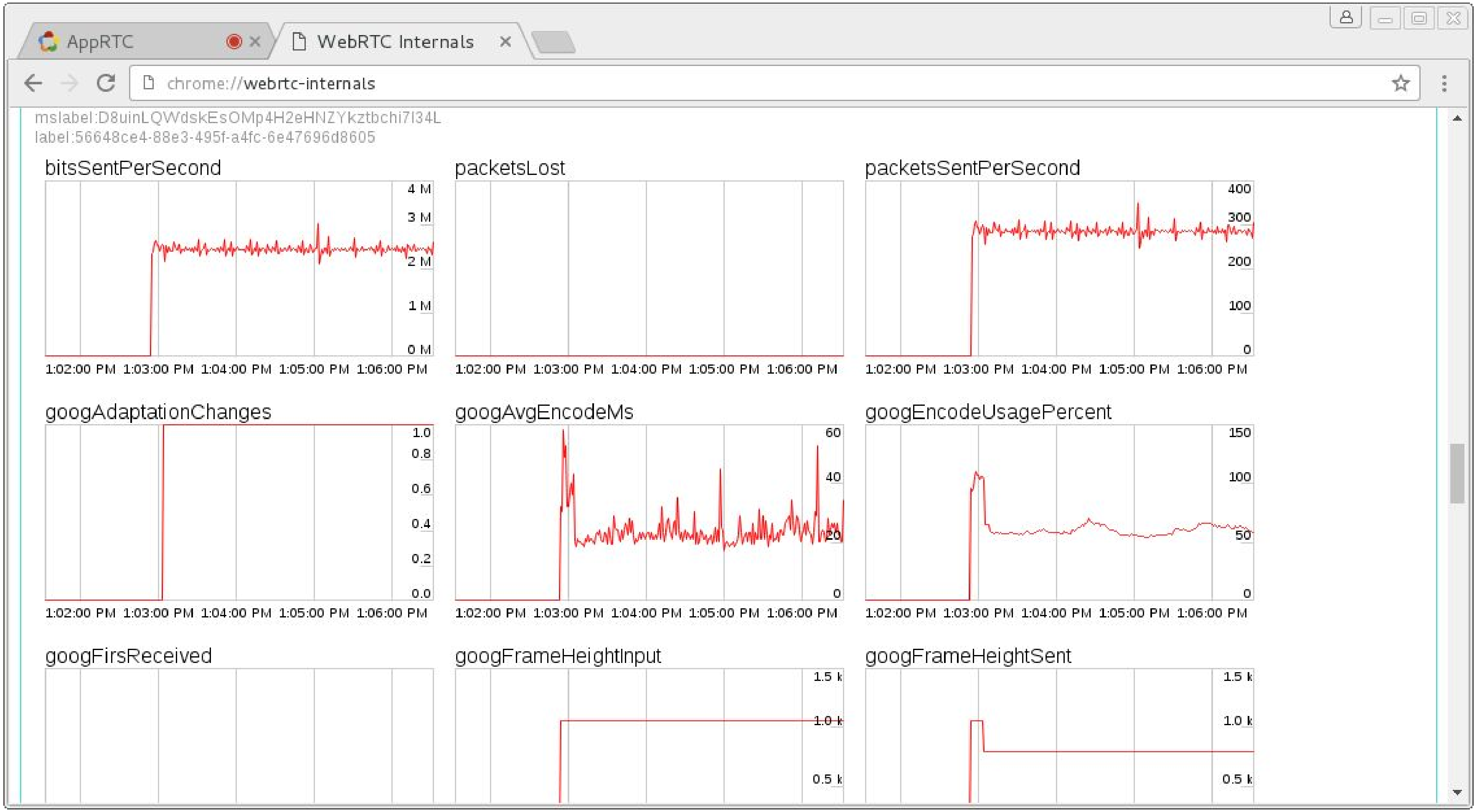
1 Click here

2. Start the call

4. End the call

5. Post process the results with `rtp_analyzer.sh`

chrome://webrtc-internals : Real-time metrics plot



CONCLUSION

The analyzed congestion control algorithm:

1. Tracks the link capacity and provides fairness in the presence of concurrent flows
2. Keeps queuing delays small

Future work

1. Move all the logic at the sender
2. Push forward the standardization

Politecnico di Bari
Dipartimento di Ingegneria Elettrica e dell'Informazione

C3LAB
Control of Computing and Communication Systems Lab

THANK YOU!

Luca De Cicco
luca.decicco@poliba.it