Palermo Congestion Control Sender Side

IETF 113 – Pre-IETF LAC

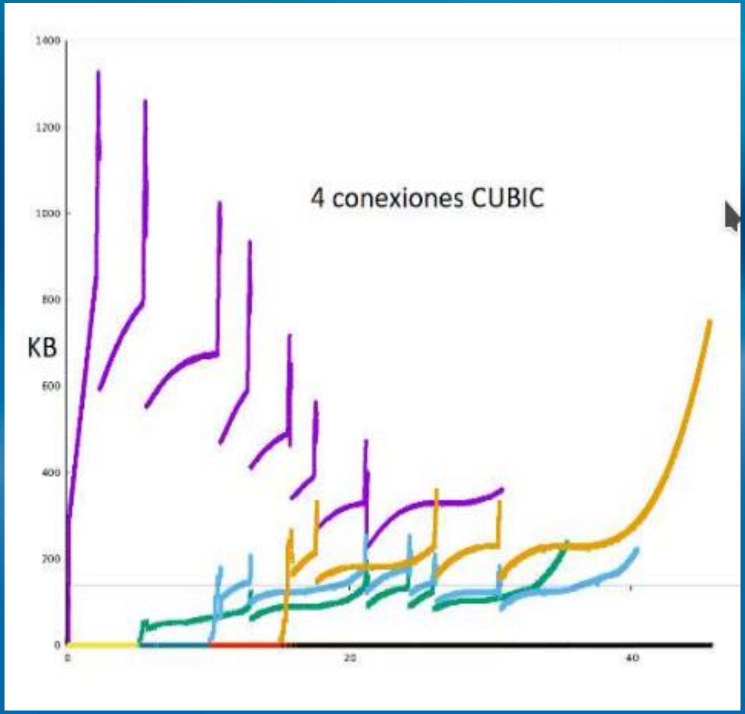
Alejandro Popovsky <apopov@palermo.edu>
Gustavo Muzzillo <gmuzzi@palermo.edu>
Universidad de Palermo, 2022

(This work was financed by FRIDA – LACNIC)

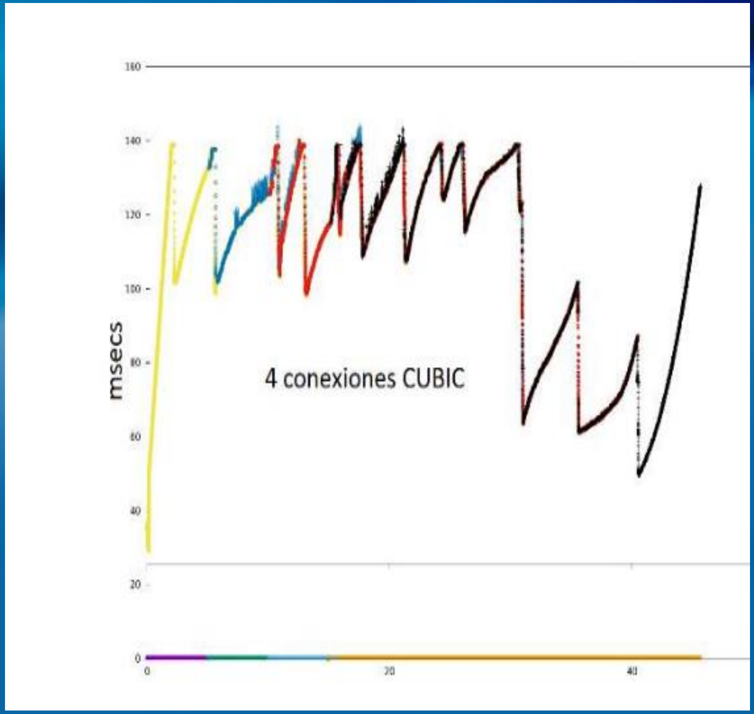
TCP congestion control Most common algorithms

- Goals:
  - Maximize throughput
  - Minimize losses
- Based on loss detection
- Reno / Cubic

TCP congestion control
Most common algorithms PROBLEMS



Cubic vs Cubic inflight data



Cubic vs Cubic round trip times

First Cubic connection bloats buffer, and gets most of the capacity till several recovery rounds

Buffer bloat effects

- Increased latency for connections sharing the bottleneck with long transmissions.
- Huge effect on transaction-oriented connections: web pages, requests-responses, etc.
- Affecting other users’ connections and also same user’s connections.

Bottleneck Sharing

Individual connection share of Capacity = Individual connection percentage of queue occupation

=> Little incentives for buffer bloat prevention

Traditional Latency aware congestion control

- Examples: LEDBAT, VEGAS, VENO, TCP-LP, …
- Sharing performance: “Less than best effort Congestion Control” (LBE)
- Alternatives to LBE: adaptive behavior

Palermo Bottleneck feedback

Goal: estimate the

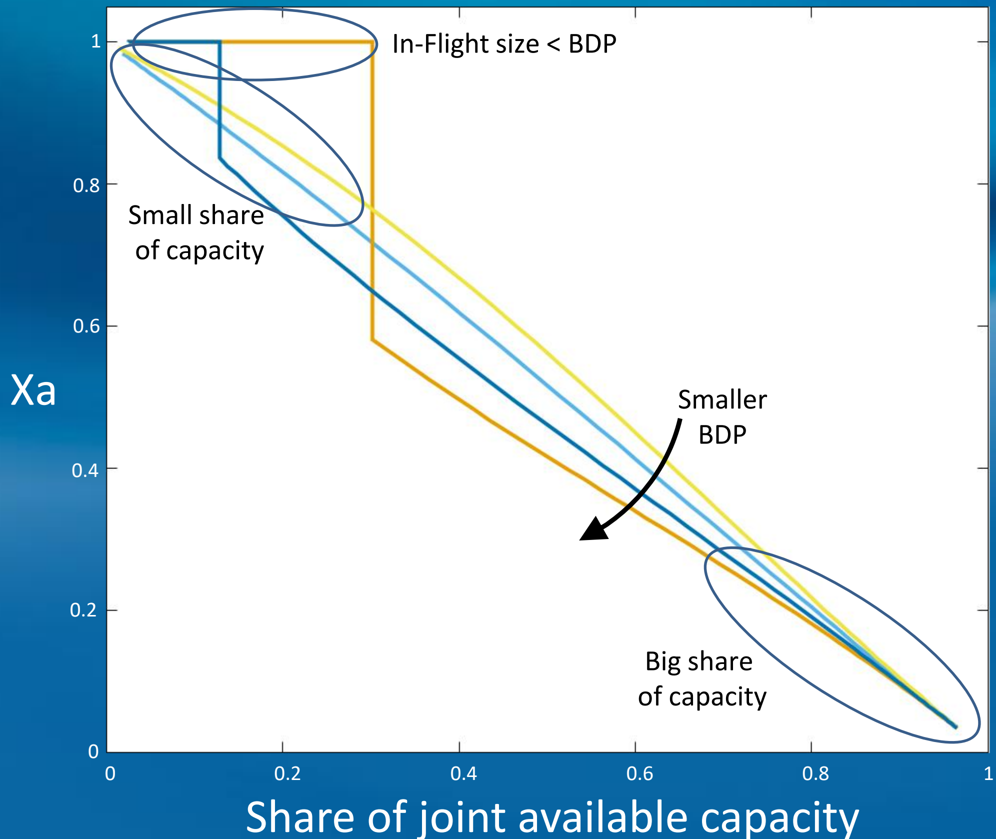
share of the joint available capacity

Proposed variable:

Proportional rate (Ra) response to In-flight size (Ca) variations

$$Xa = \left(\frac{\Delta Ra}{\Delta Ca}\right)\left(\frac{Ca}{Ra}\right)$$

Estimating Bottleneck share with Xa



Exclusive user of bottleneck:

In-Flight size<BDP => Xa=1

In-Flight size>BDP => Xa=0

Shared bottleneck:

Xa ≈ (1 – share of capacity)

Palermo Algorithm

- Maintain:
  - Optimize troughput
  - Minimize loss
- Add:
  - Minimize latency
  - Fair sharing

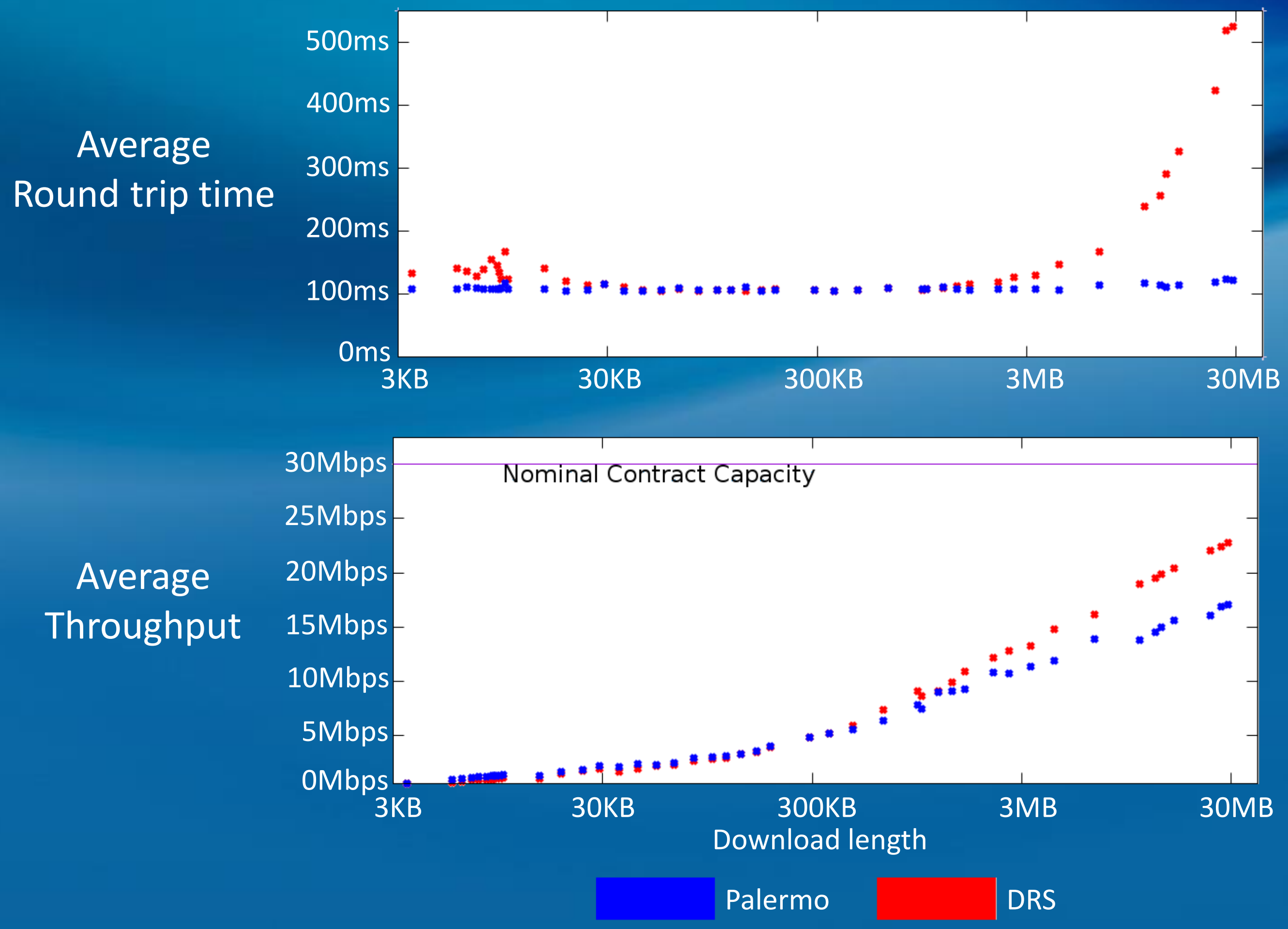
- Adaptive behavior:
  - Buffer bloat prevention if posible
  - Else revert to regular congestión control

Palermo Receiver Congestion Control

- Developed in 2016, presented IETF 95.
- Opportunity observed in CABASE IXP's:
  - Flow controlled traffic without dynamic receive window (DRS), sometimes gets same throughput as congestión controlled traffic, but with smaller latency.
- Thoroughly tested in University of Palermo proxies for incoming traffic.

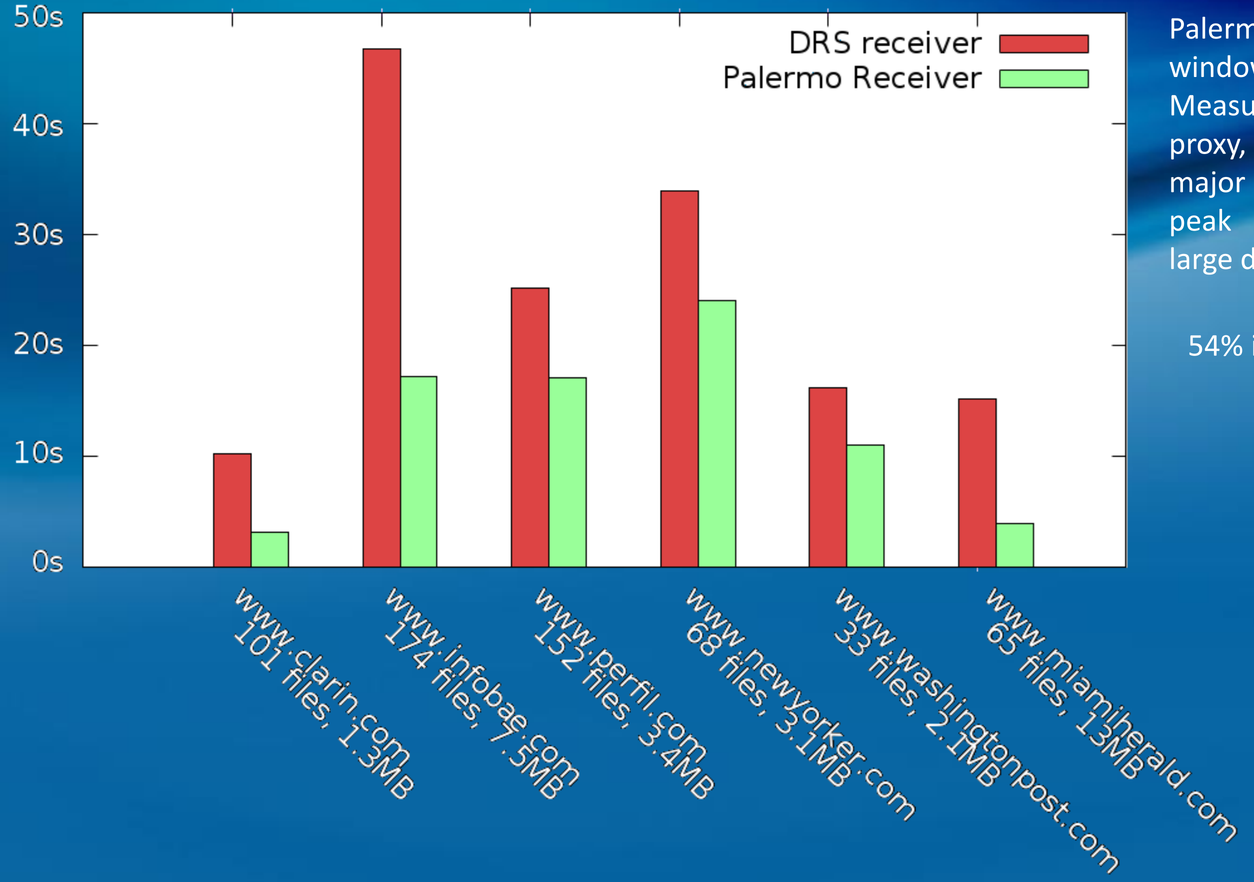
Palermo Receiver Side Performance

Palermo versus DRS receiver window control. Measurements at university proxy, averaging over several Centos mirrors



Palermo Receiver Side Performance

Total download time



Palermo versus DRS receiver window control.
Measurements at university proxy, Downloading from major newspapers during peak hours (in parallel to large downloads)

54% improvement

Receiver vs Sender Congestion Control

- Receiver side:
  - Optimize incoming traffic.
  - Current limitation uncertainties
    Application , Flow ctrl, or sender cwnd.
  - Round trip time measurements unaccuracy.
  - Need to slow sender in order to gain control

- Sender Side
  - Optimize outgoing traffic
  - Better knowledge of current limitation
  - Better measurements of round trip times

- Both sides
  - Need to wait for adaptations to show effects in feedback variable Xa

Palermo Algorithm Details

- Available capacity not jointly reached
  => regular cwnd growth

- Available capacity jointly reached
  (number of conns sharing the bottleneck: unknwon)
  - but having a small percentage =>regular growth
  - but having 100% or high percentage => oscillate

- Adaptation criteria:
  - algorithm obtains a latency that grows with the number of sharing conns. So no gain is achievable when conns count exceed a limit, so revert to regular aggresive (reno-cubic) congestión control.

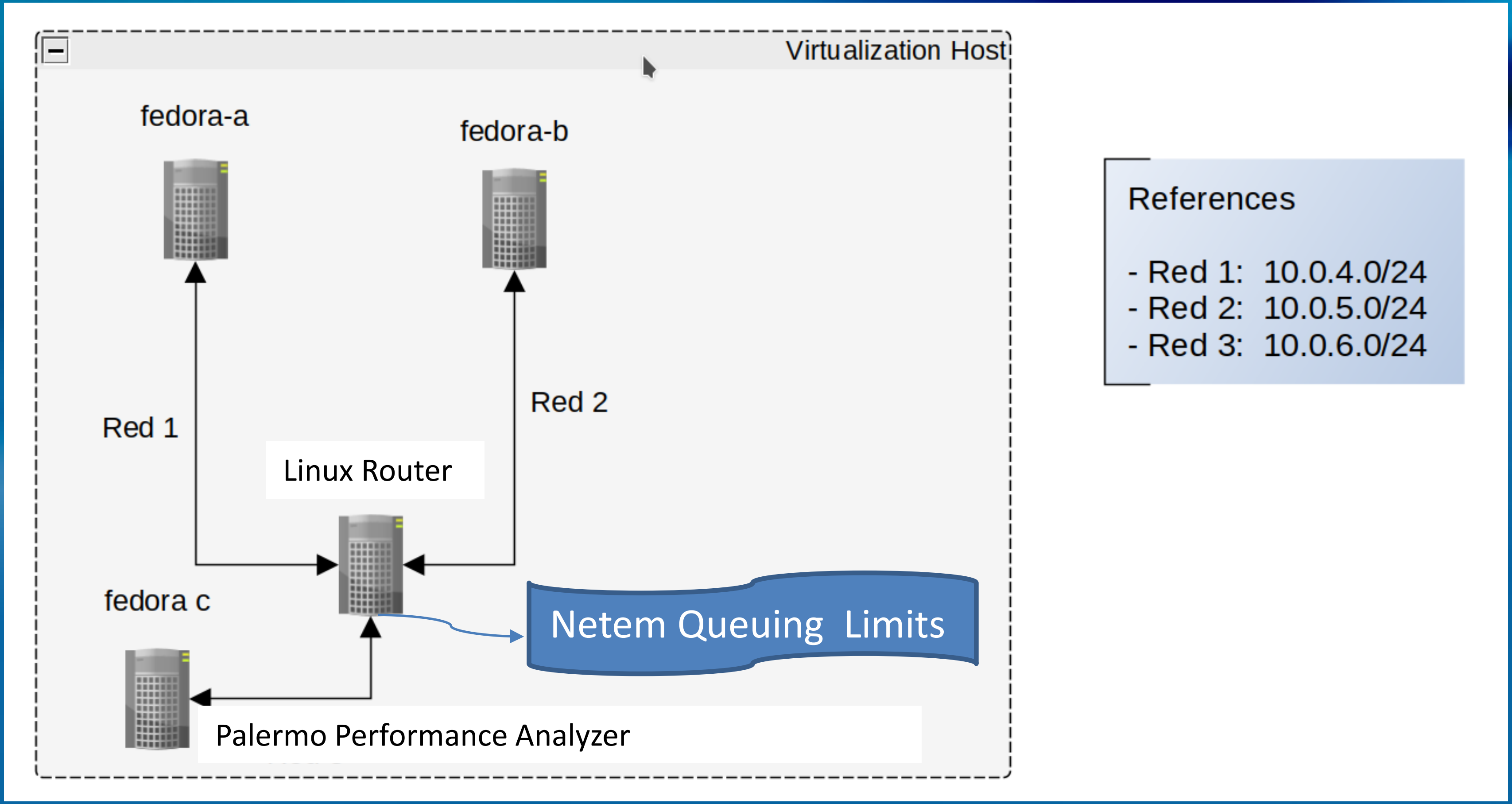
TCP congestion control
Palermo sender Architecture

- Developed for Linux Kernels 5.9, and above.
- Independent Dynamic Kernel Module (LKM, DMKS)
- Tested for X86_64 architecture extendable to other architectures like ARM, PPC, etc.

Test Objectives

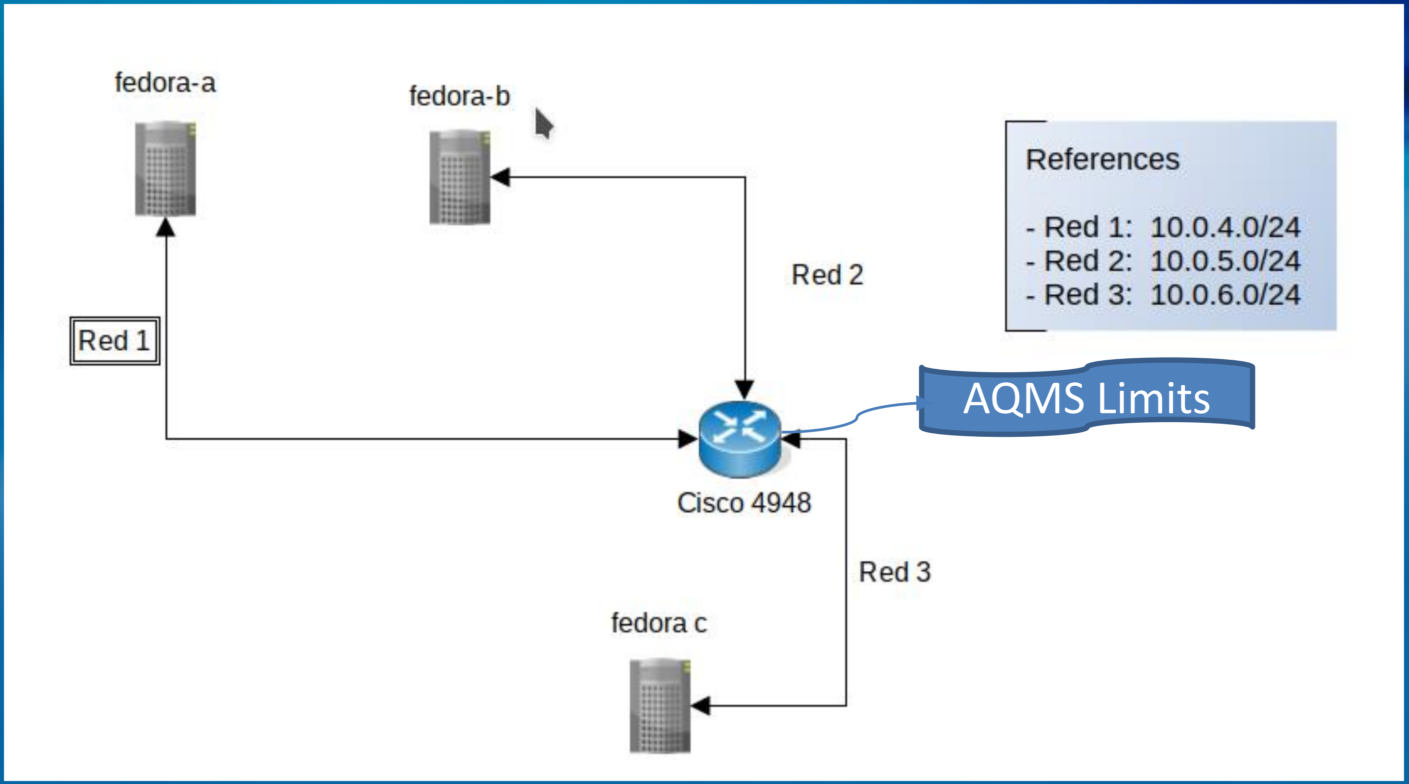
- Check throughput maximization when connection alone at the bottleneck
- Check bufferbloat prevention when sharing bottleneck with well behaved connections
- Check fair sharing of bottleneck capacity when sharing bottleneck with well behaved connections.
- Check performance when sharing bottleneck with bad-behaved (traditional) connections.
- Compare with state-of-the-art competing algotithms (BBR)
- Check large scale deployment in datacenters.

Testbench Infrastructures



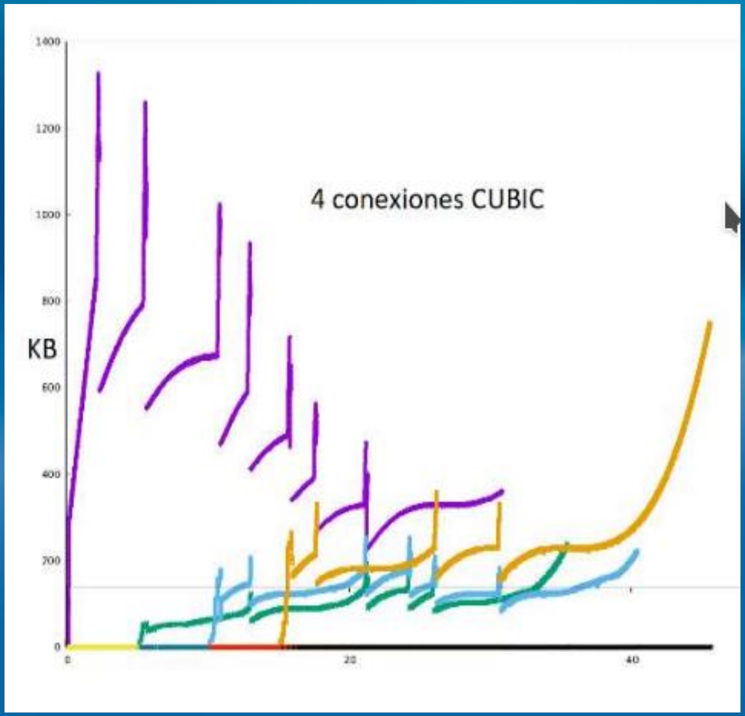
TestBench 1

Testbench Infrastructures (cont.)

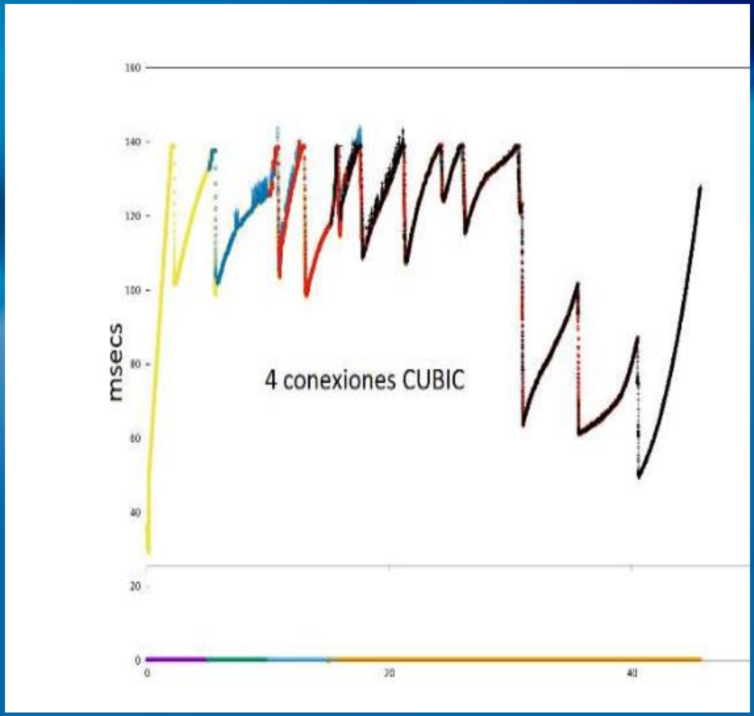


TestBench 2

CUBIC vs CUBIC tests



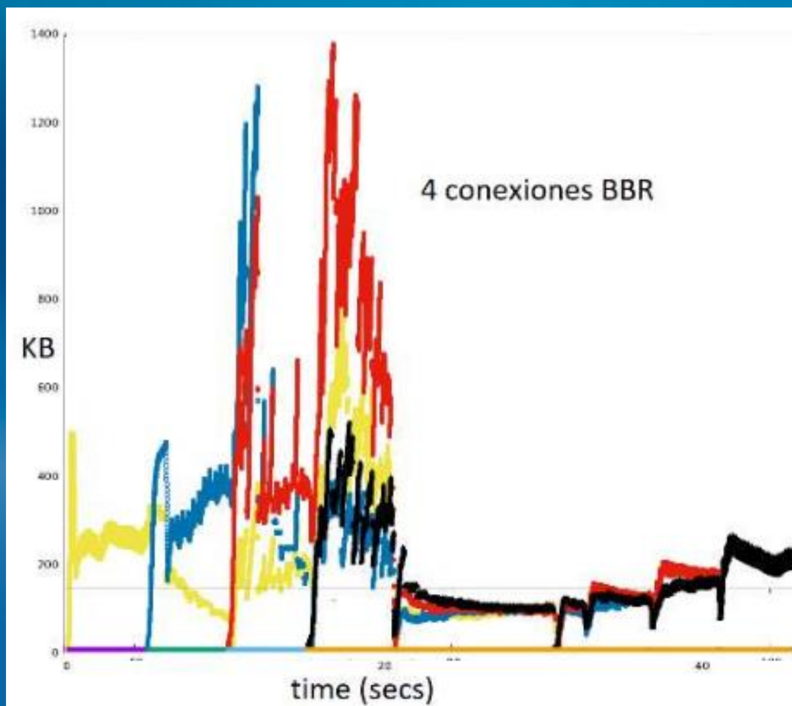
Cubic vs Cubic inflight data



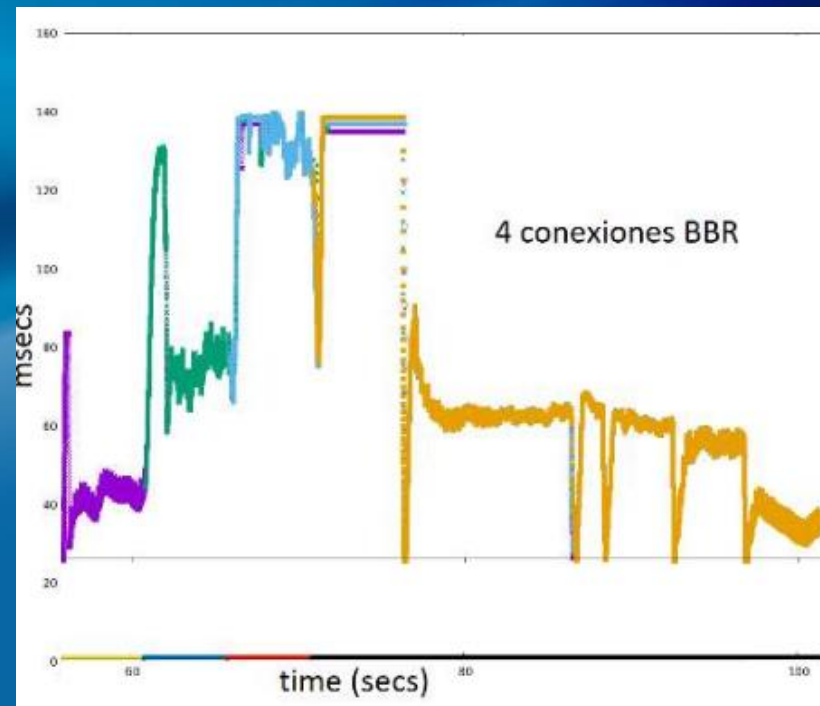
Cubic vs Cubic round trip times

First Cubic connection bloats buffer, and gets most of the capacity till several loss-recovery rounds

BBR vs BBR tests

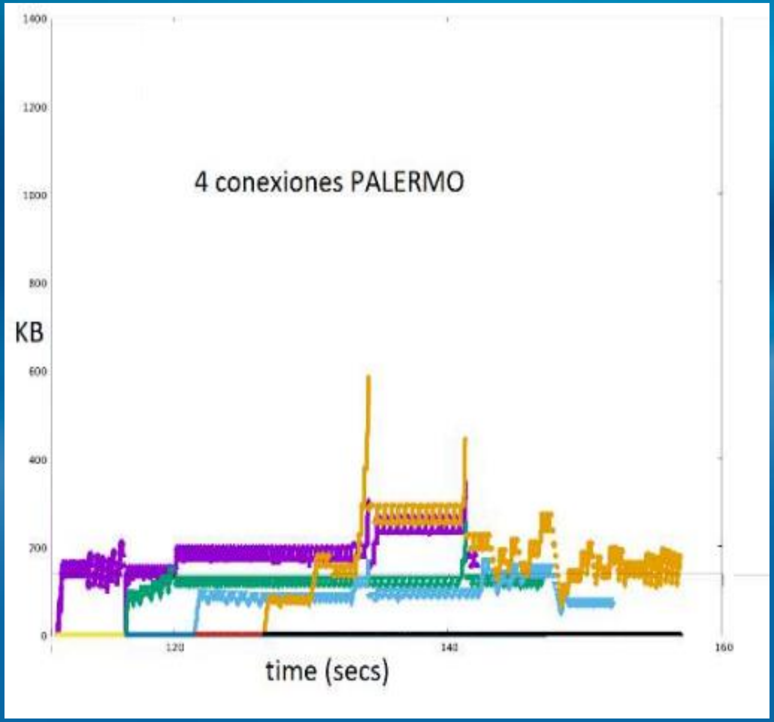


Bbr vs Bbr inflight data

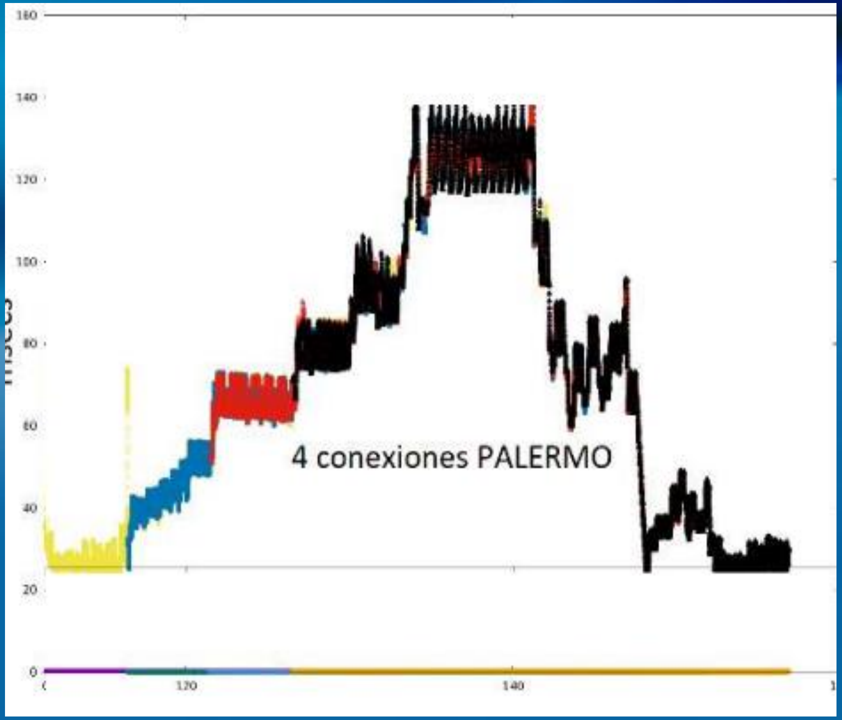


BBR vs BBR round trip times

Palermo vs Palermo test results



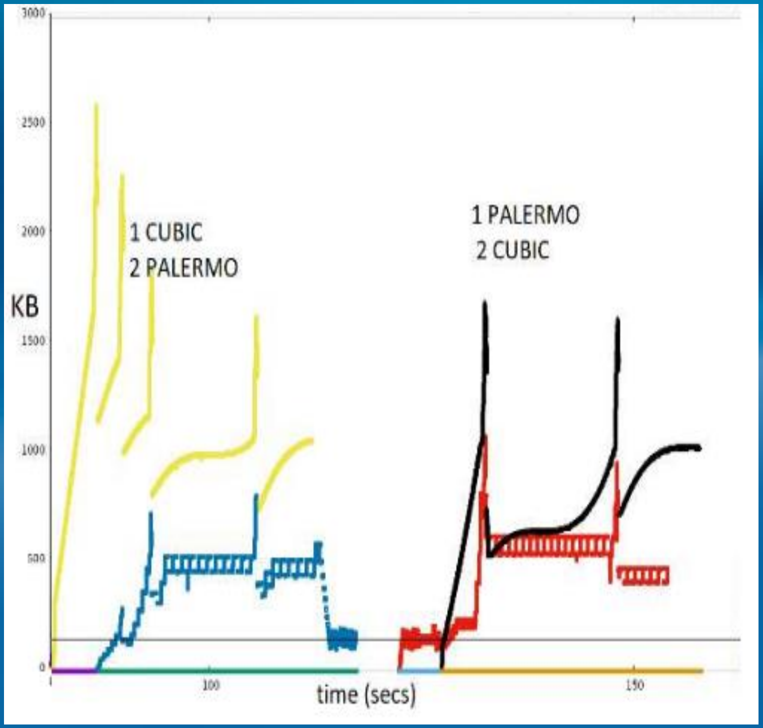
Palermo vs Palermo inflight data



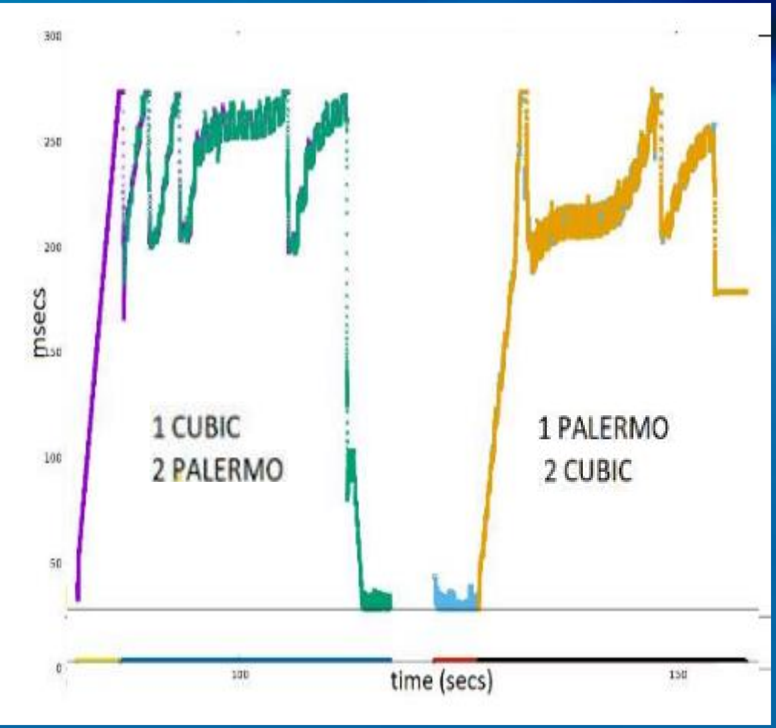
Palermo vs Palermo round trip times

UP Universidad de Palermo

Palermo vs Cubic test results

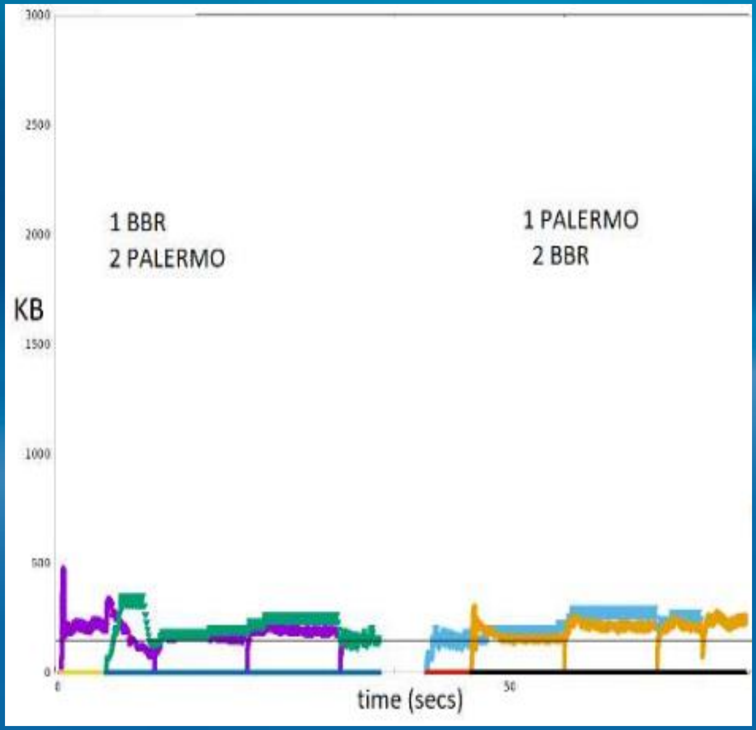


Palermo vs Cubic inflight data

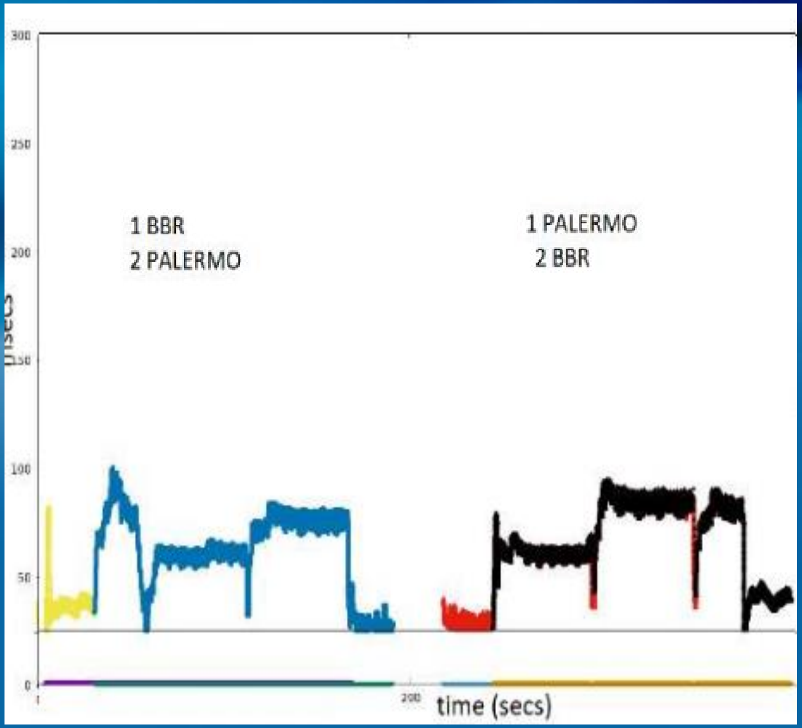


Palermo vs Cubic round trip times

Palermo vs BBR test results

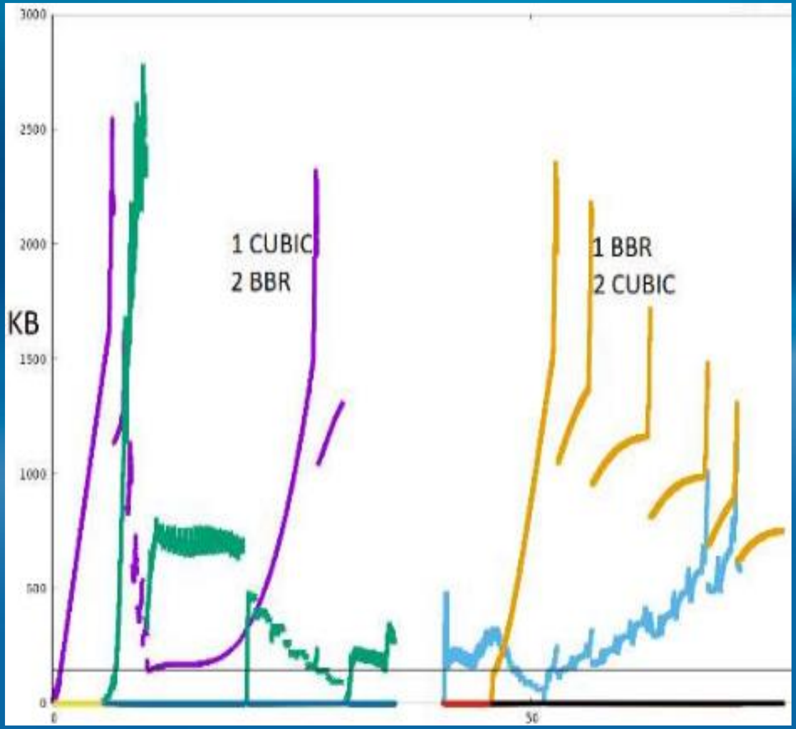


Palermo vs bbr inflight data

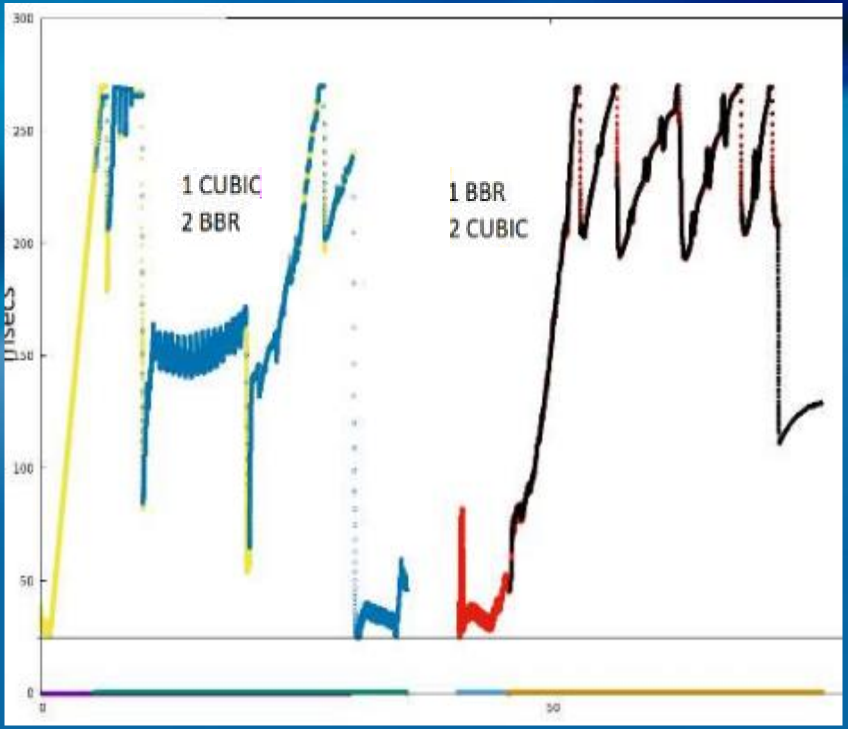


Palermo vs bbr round trip times

CUBIC vs BBR test results



Cubic vs bbr inflight data



Cubic vs bbr round trip times

Conclusions and Future Work

- Proposed algorithm:
  - As a valid option for being used by organization's servers to improve their performance on outgoing traffic.
- Next:
  - Explore robustness and variants
  - Upcoming publication
  - Large scale tests in datacenters
  - Proposal for Kernel std distribution

For more information, or source code:
https://www.palermo.edu/ingenieria/ingenieria-telecomunicaciones/control-de-congestion-palermo.html
apopov@palermo.edu