NADA Update: Algorithm, Implementation, and Test Case Evaluation Results

draft-zhu-rmcat-nada

Xiaoqing Zhu, Michael Ramalho, Charles Ganzhorn, Paul Jones, and Rong Pan

IETF 90, Toronto, Canada

2014-07-24

Outline

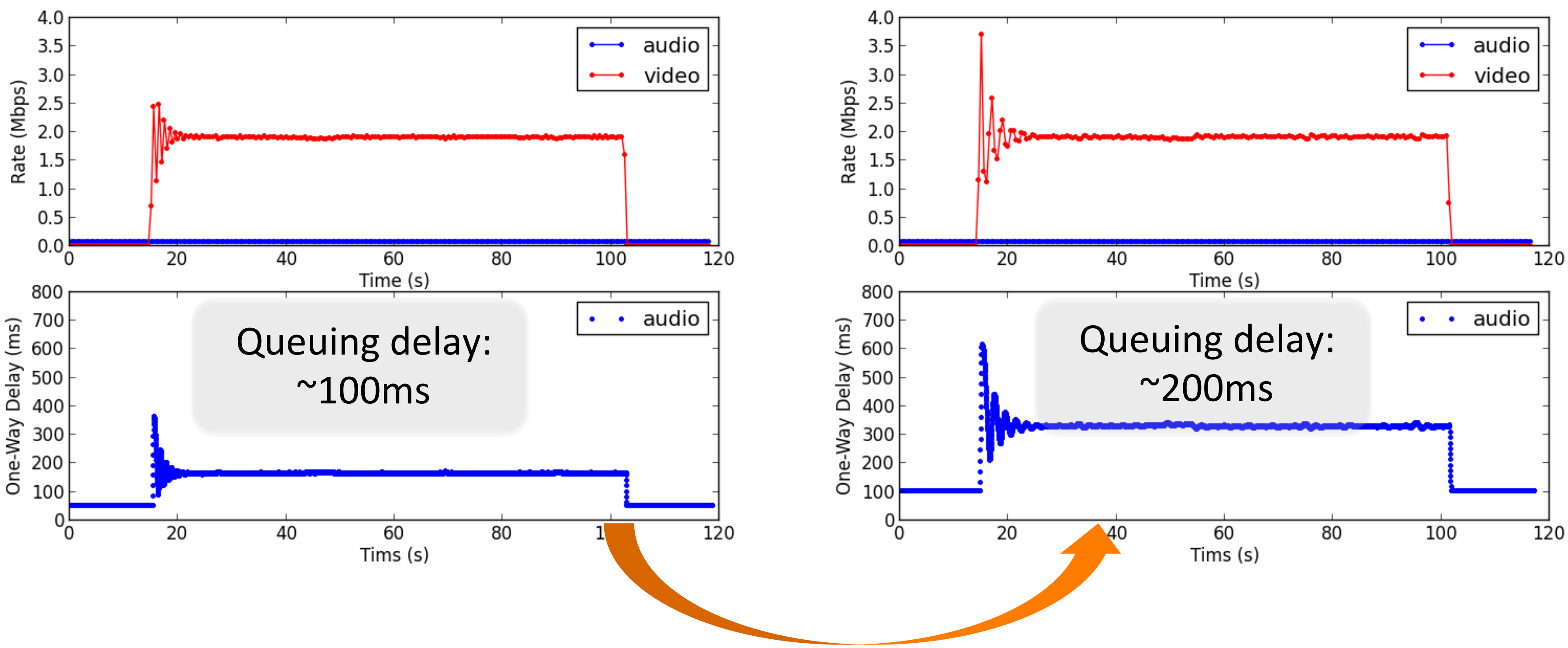
- Revised algorithm
- Simulation-based test case evaluation
- Testbed status

Issues with Currently Documented Algorithm (as in draft-zhu-rmcat-nada-03)

- Relies on per-packet feedback from receiver:
  - May incur too much overhead using RTCP messages
- Uses one-way delay (queuing + propagation) as a primary form of congestion indication:
  - May not work well without well-synchronized sender and receiver clocks over Internet
  - At equilibrium, queuing delay and forward delay scale linearly with forward propagation delay

Impact of Forward Propagation Delay on Queuing: Simplified Algorithm in Testbed

Forward Propagation Delay = 50ms **Forward Propagation Delay = 100ms**



2x increase in queuing delay

How to Fix This?

- Use queuing delay instead as the primary form of congestion indication
- Gradual rate update based on periodic measurements of both the *value* and *change* of queuing delay
- Decouple DC and high-frequency gains in the control loop to ensure stability while maintaining weighted bandwidth sharing at steady state

Revised NADA Receiver Behavior

- Obtain per-packet observations:

$$x_n = t_{r,n} - t_{s,n}$$

$$\mathbf{1}_M := \begin{cases} 0, & \text{no marking} \\ 1, & \text{w/ marking} \end{cases}$$

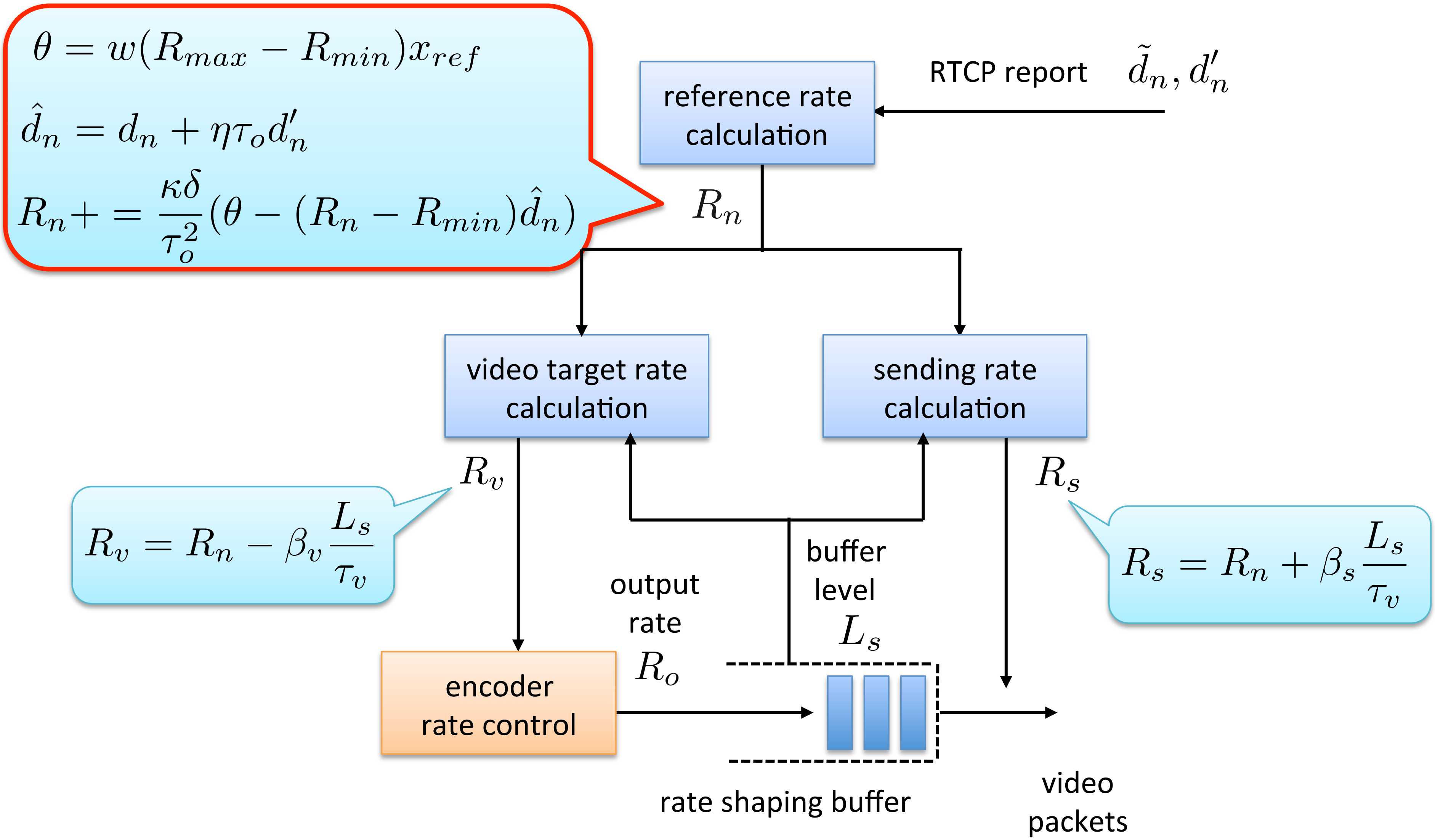
$$\mathbf{1}_L := \begin{cases} 0, & \text{no loss} \\ 1, & \text{w/ loss} \end{cases}$$

- Calculate queuing delay: $d_n = x_n - d_f$ ← Forward propagation delay

- Calculate delay derivative: $d'_n = \dfrac{d_n - d_{n-k}}{\delta}$ ← Interval between adjacent ACKs

Pending design on how to incorporate loss and marking information.

Revised NADA Sender Behavior



Digesting the Rate Update Equation

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}(\theta - (R_n - R_{min})(d_n + \eta\tau_o d'_n))$$

Digesting the Rate Update Equation

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}(\theta - (R_n - R_{min})(d_n + \eta\tau_o d_n))$$

At start-up

$$d_n = 0, d'_n = 0$$

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}\theta$$

Linear increase with a fixed slope

Digesting the Rate Update Equation

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}(\theta - (R_n - R_{min})(d_n + \eta\tau_o d_n'))$$

At start-up

$d_n = 0, d_n' = 0$

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}\theta$$

Linear increase with a fixed slope

At steady-state

$R_n' = 0, d_n' = 0$

$$R_o = R_{min} + \frac{\theta}{d_o}$$

Ensures weighted bandwidth sharing

Digesting the Rate Update Equation

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}(\cancel{\theta} - (R_n - R_{min})(\cancel{d_n} + \eta\tau_o d'_n))$$

<u>At start-up</u>

$$d_n = 0, d'_n = 0$$

$$R_n \leftarrow R_n + \frac{\kappa\delta}{\tau_o^2}\theta$$

Linear increase with a fixed slope

<u>At steady-state</u>

$$R'_n = 0, d'_n = 0$$

$$R_o = R_{min} + \frac{\theta}{d_o}$$

Ensures weighted bandwidth sharing

<u>Around equilibrium</u>

$$R_n \approx R_o, d_n \approx d_o$$

$$R'_n = -\frac{\kappa\eta}{\tau_o}(R_n - R_{min})d'_n$$

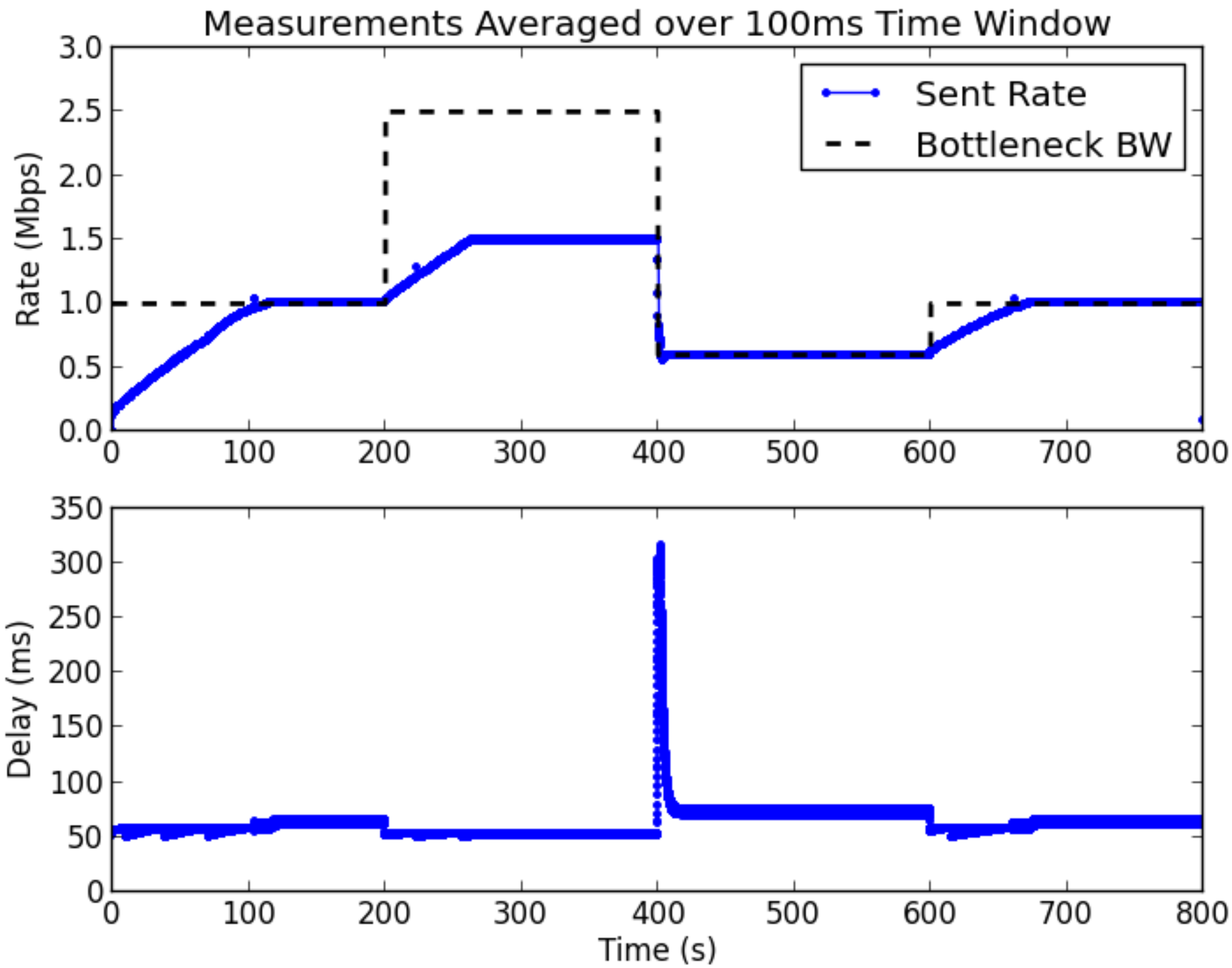
change in rate scales with change in delay

Simulation Evaluations Based on draft-sarker-rmcat-eval-test-01

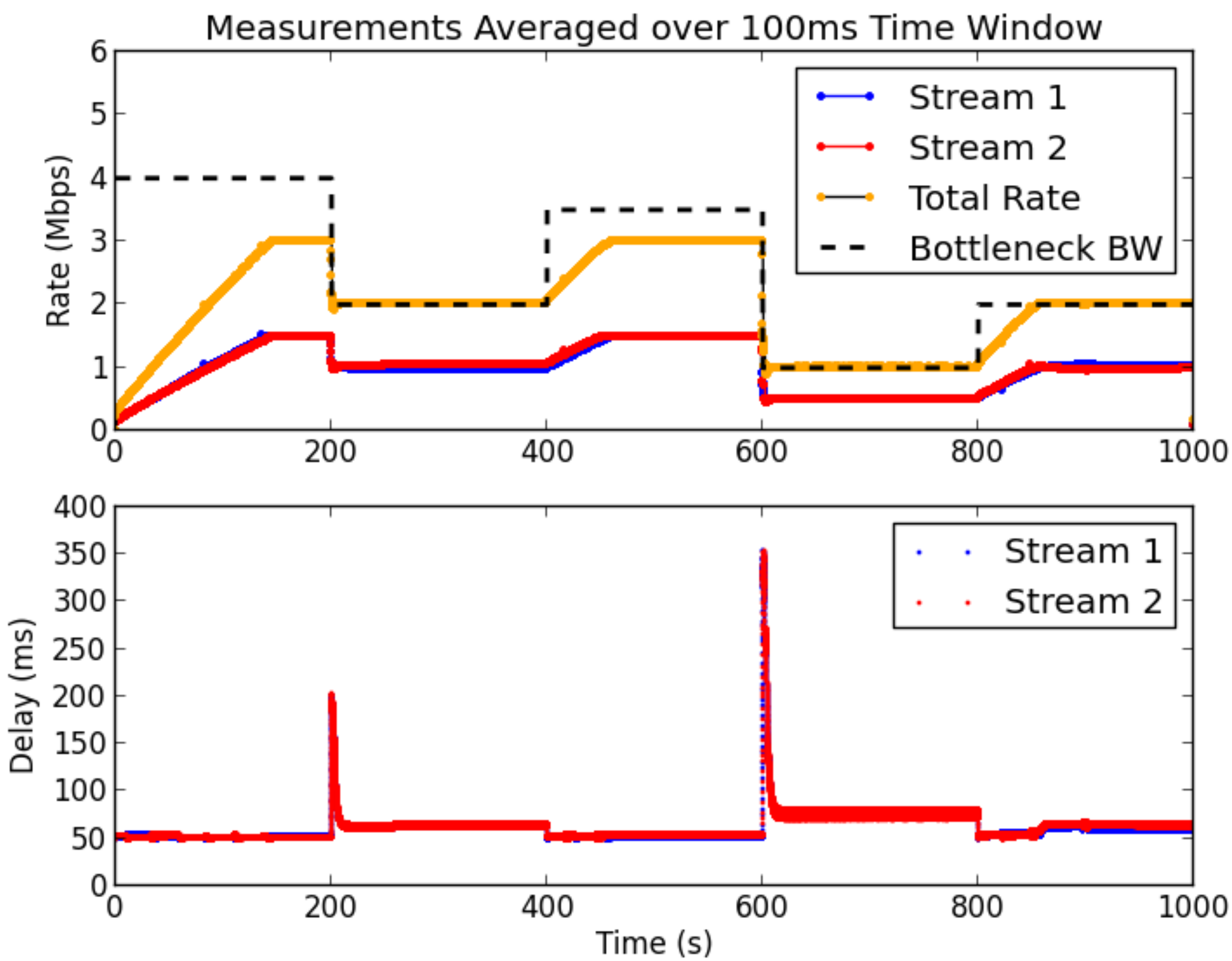
- Default path setting:
  - Forward propagation delay: 50ms
  - Bottleneck queue depth: ~300ms

- Default algorithm parameters:
  - ACK interval: once per 20 packets
  - MTU size: 1000 bytes
  - Rate range: 150 Kbps ~ 1.5 Mbps
  - Scaling parameters: $\kappa = 0.2, \eta = 5.0$
  - Upper bound on RTT: $\tau_o = 500 \text{ ms}$
  - Reference delay: $x_{ref} = 10 \text{ ms}$

Variable Available Capacity w/ Single RMCAT Flow

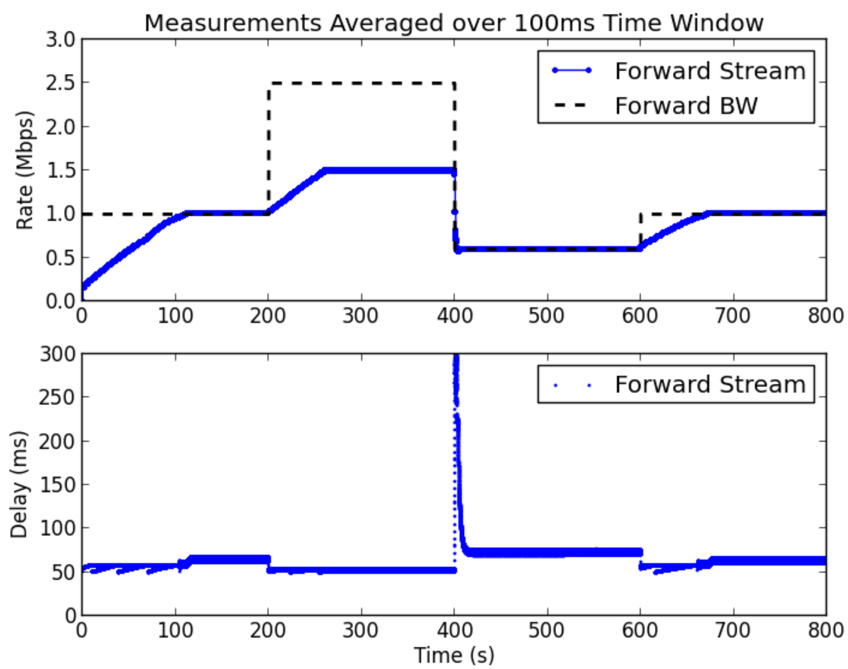


Variable Available Capacity w/ Multiple RMCAT Flows

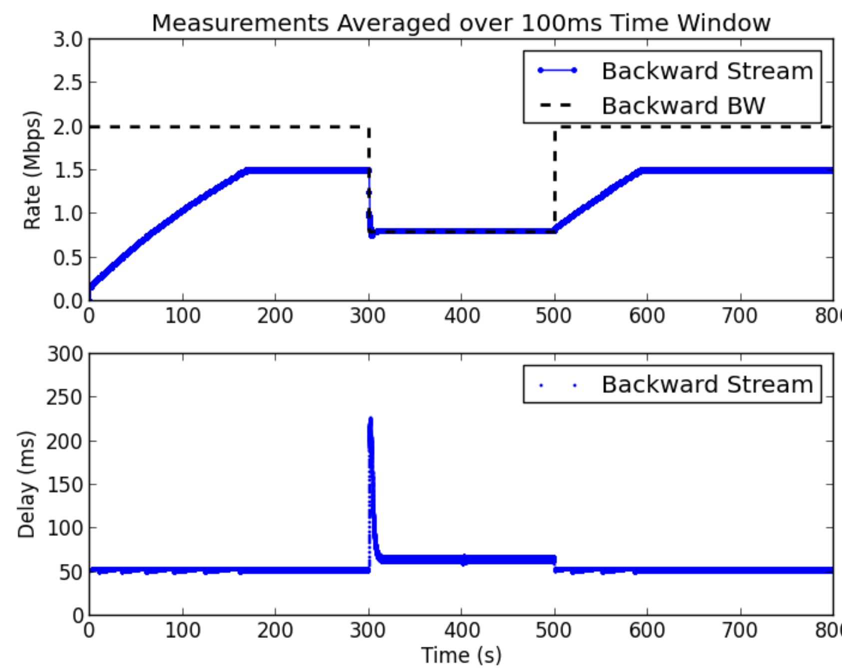


Congested Feedback w/ Bi-directional RMCAT Flows:

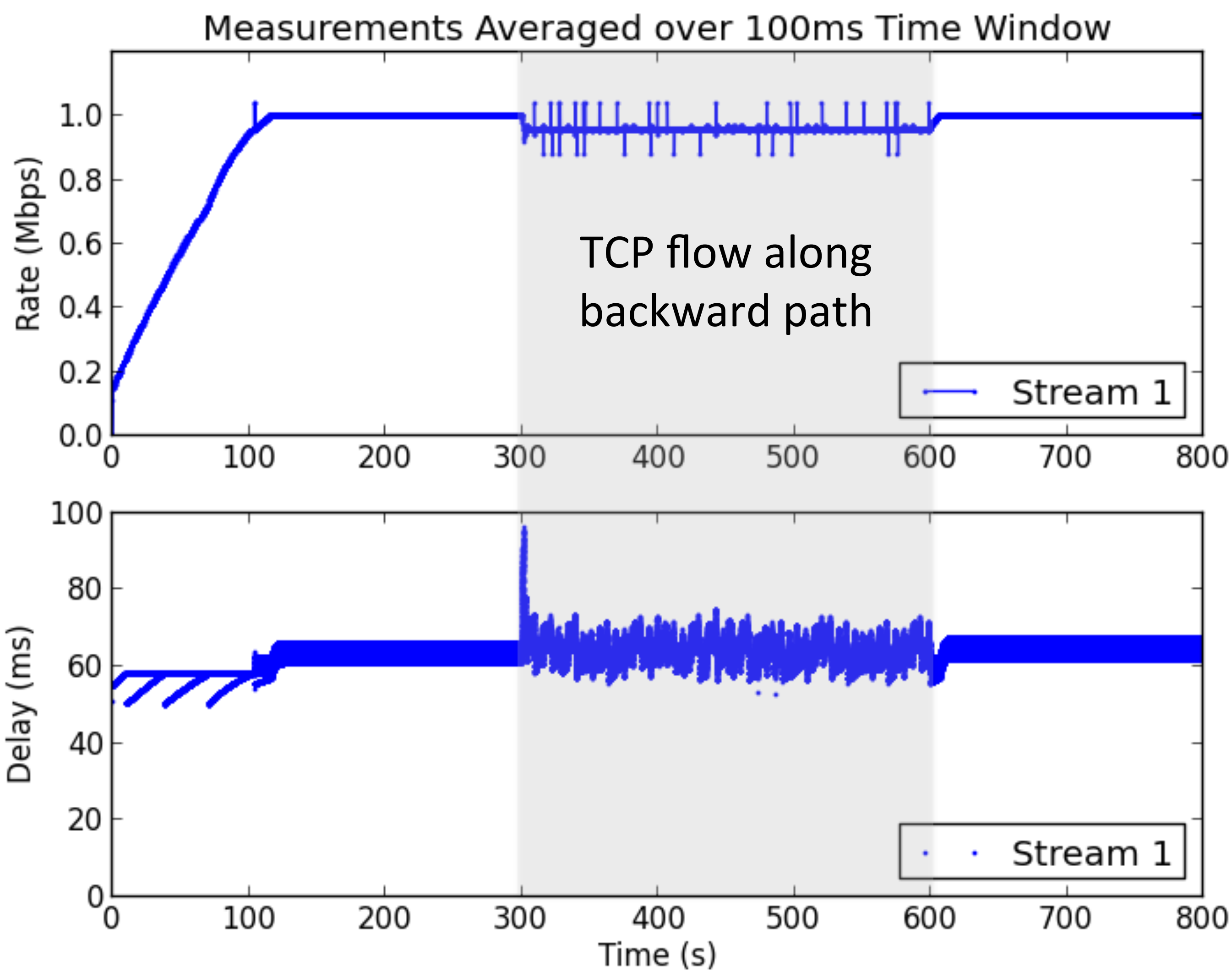
Forward Direction



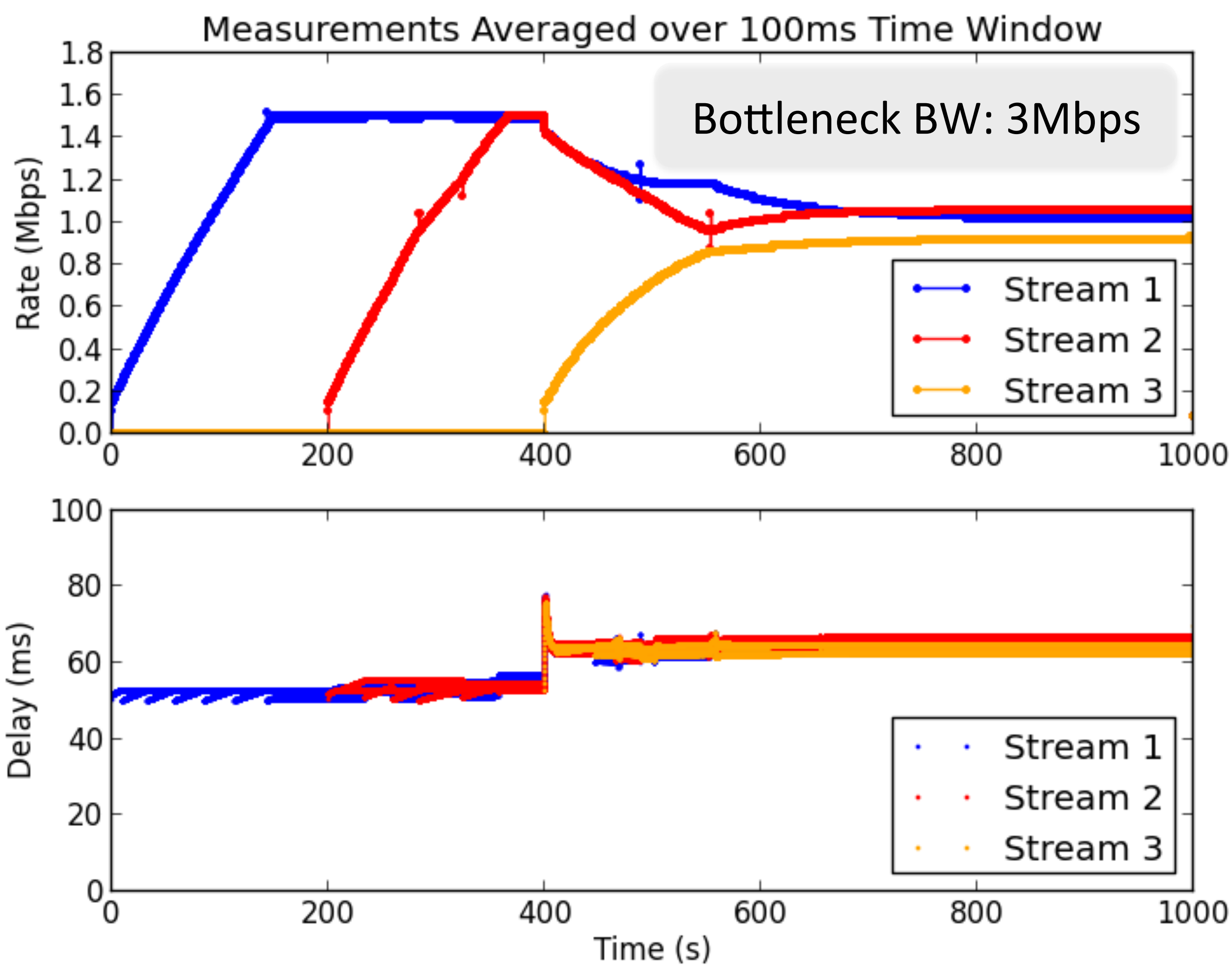
Backward Direction



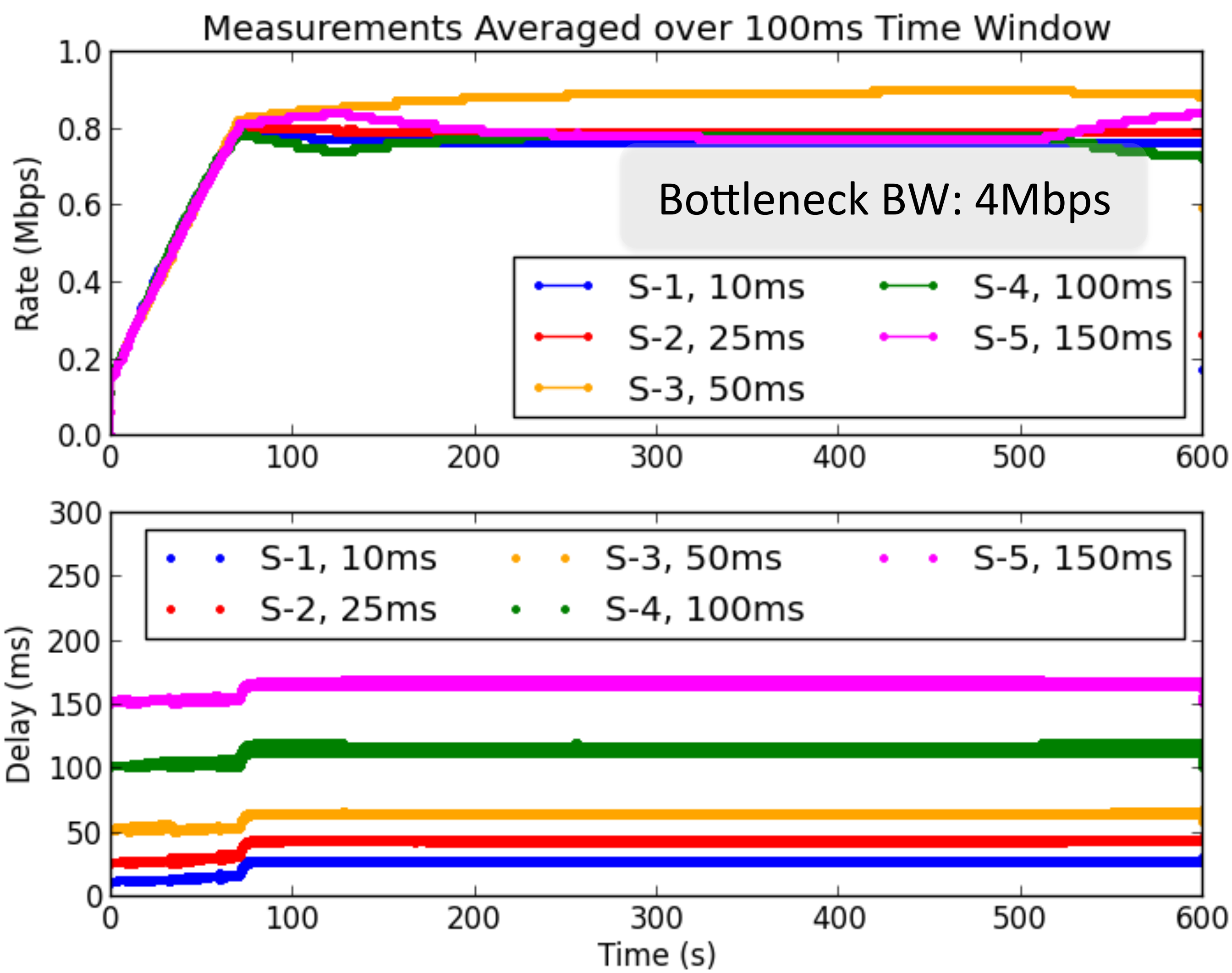
Congested Feedback w/ TCP Backward Traffic



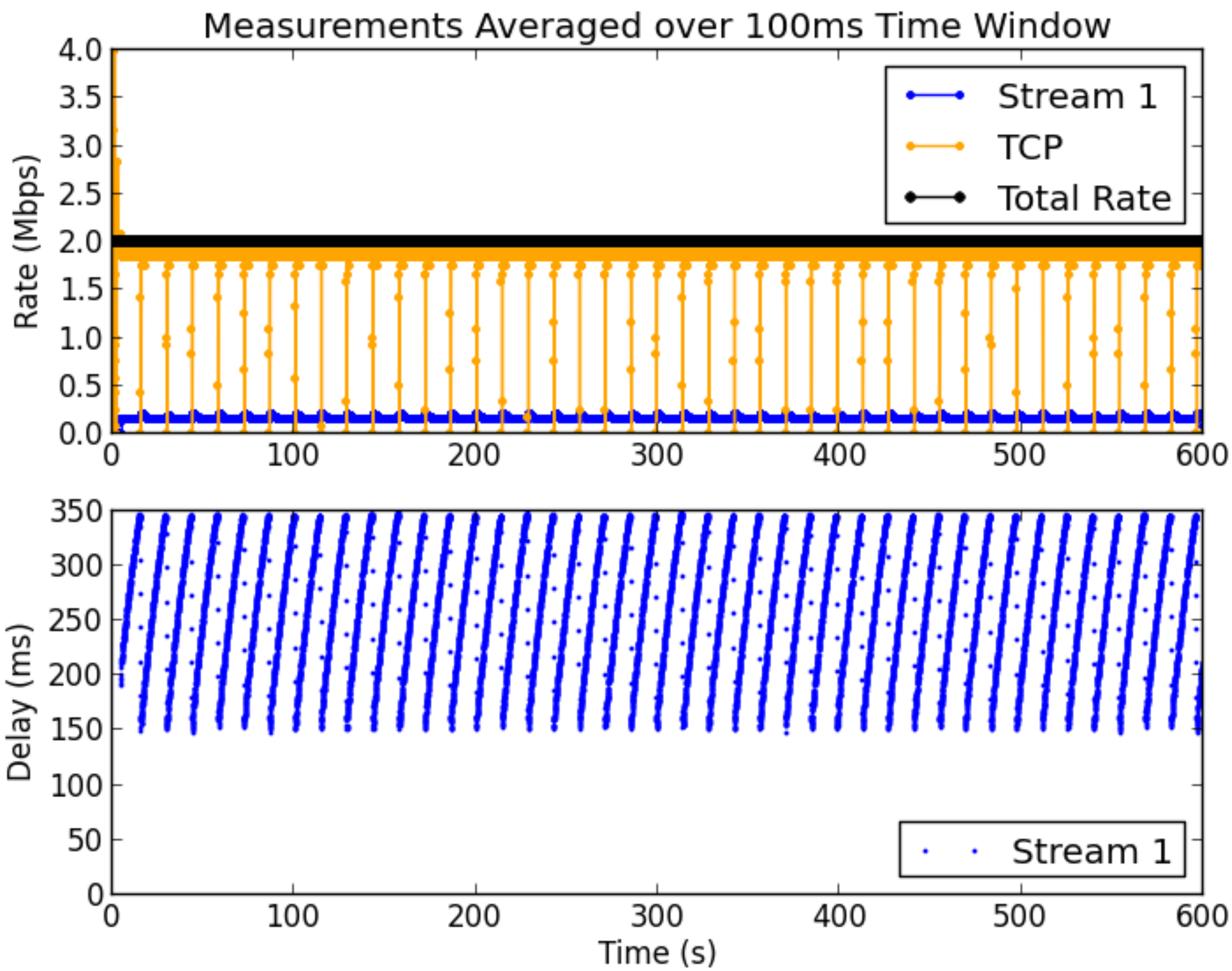
Competing Flows w/ Same RMCAT Algorithm



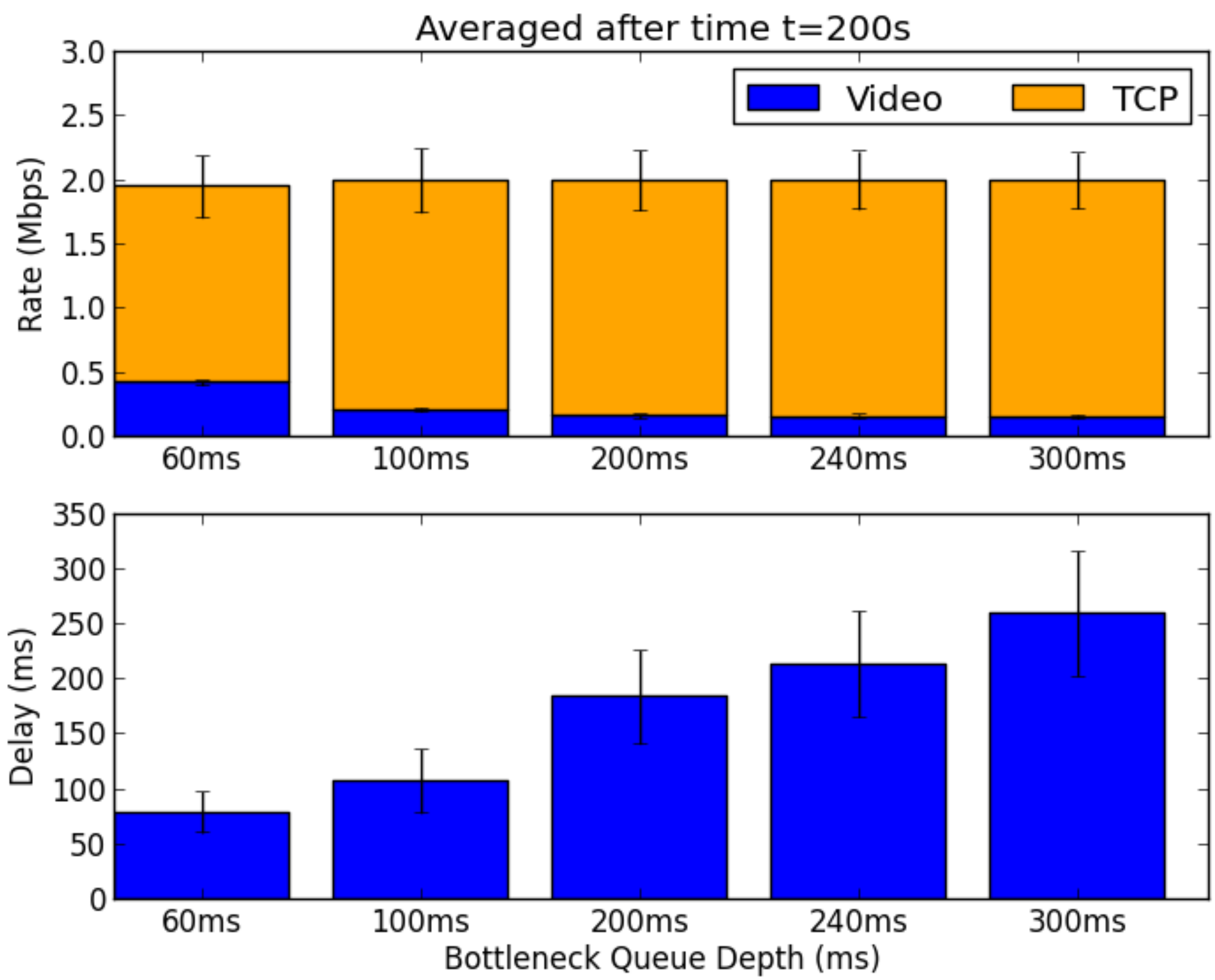
RTT Fairness



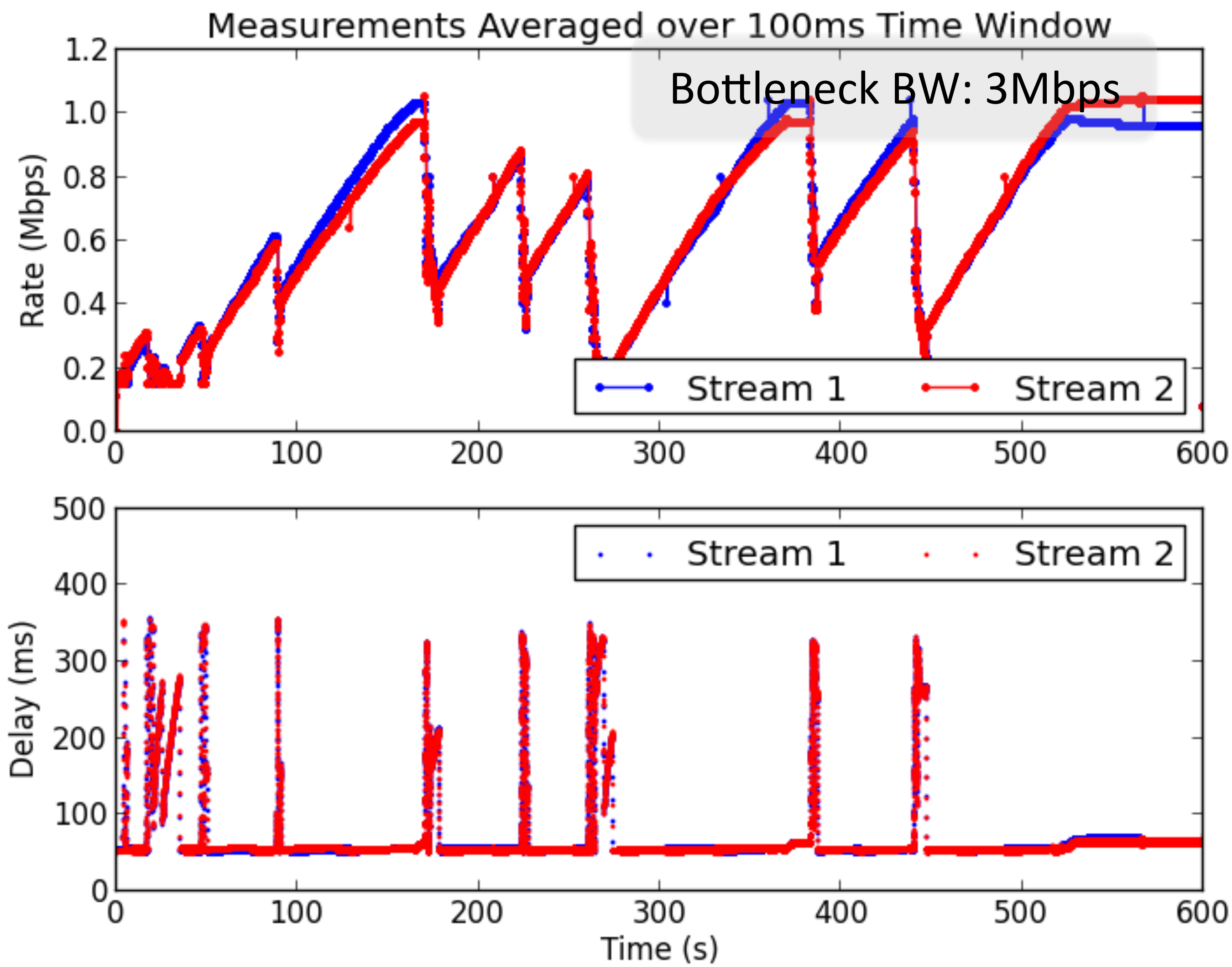
RMCAT Flow competing w/ a Long TCP Flow



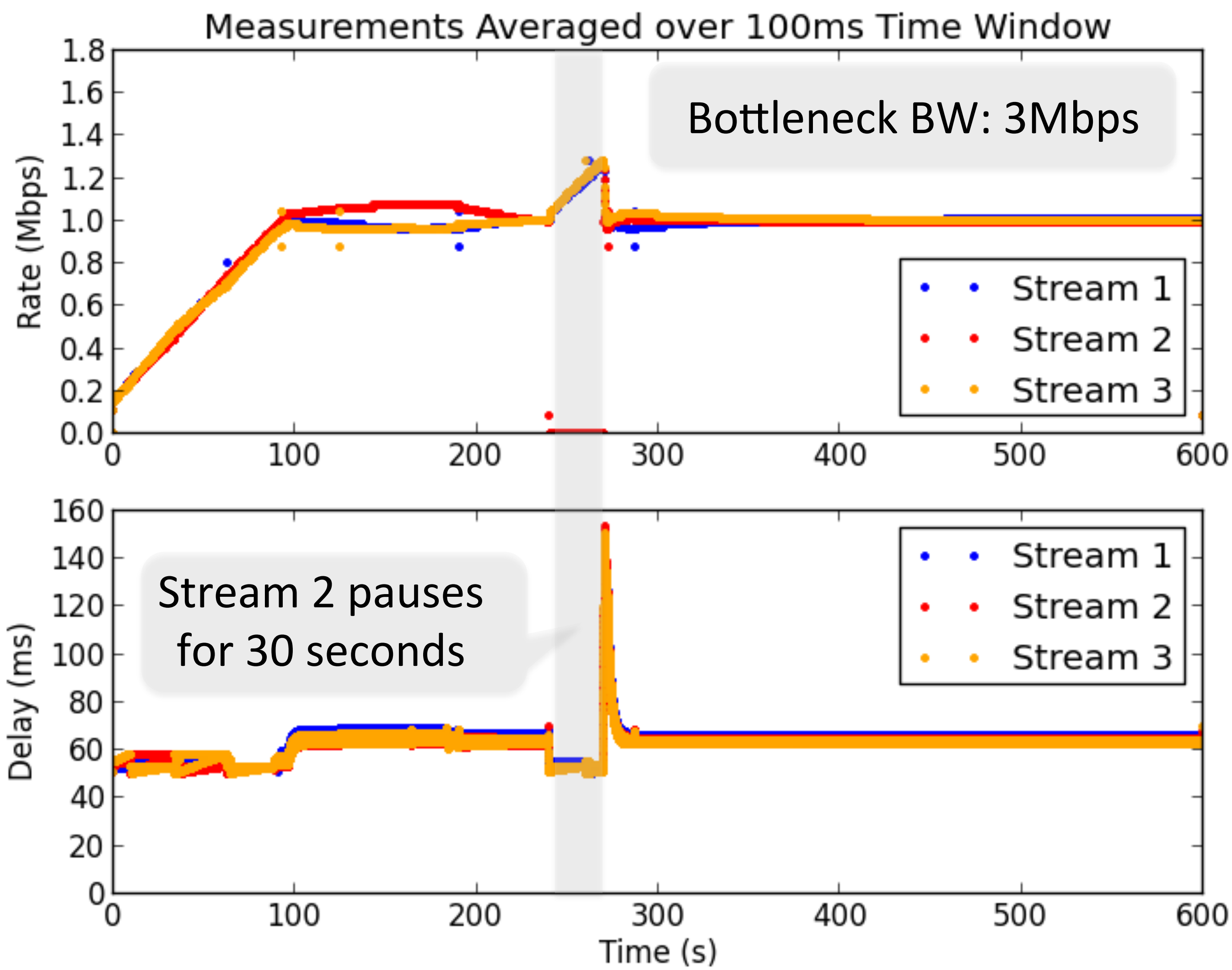
RMCAT Flow competing w/ a Long TCP Flow: Varying Bottleneck Queue Depth



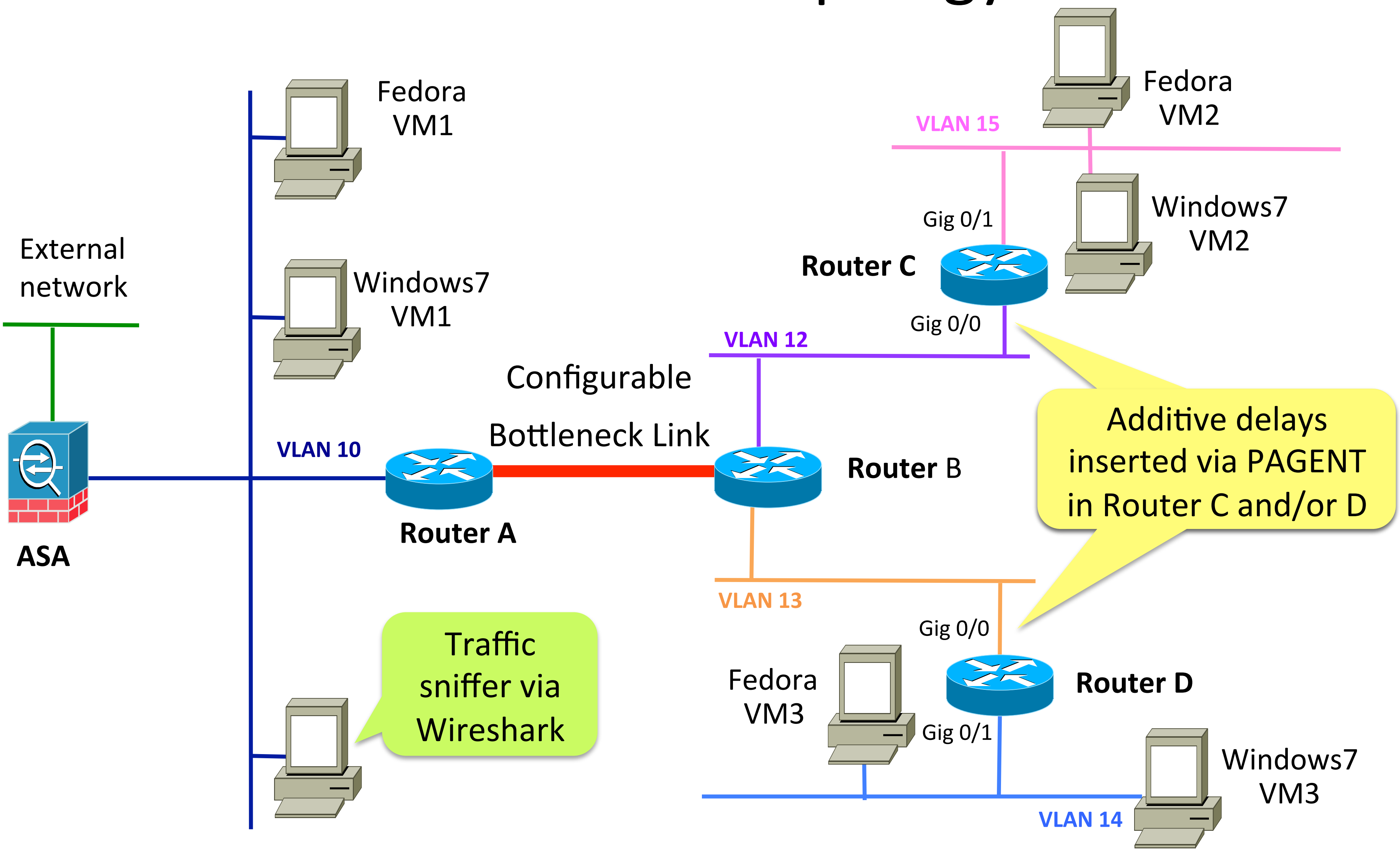
RMCAT Flow competing w/ Short TCP Flows



Media Pause and Resume

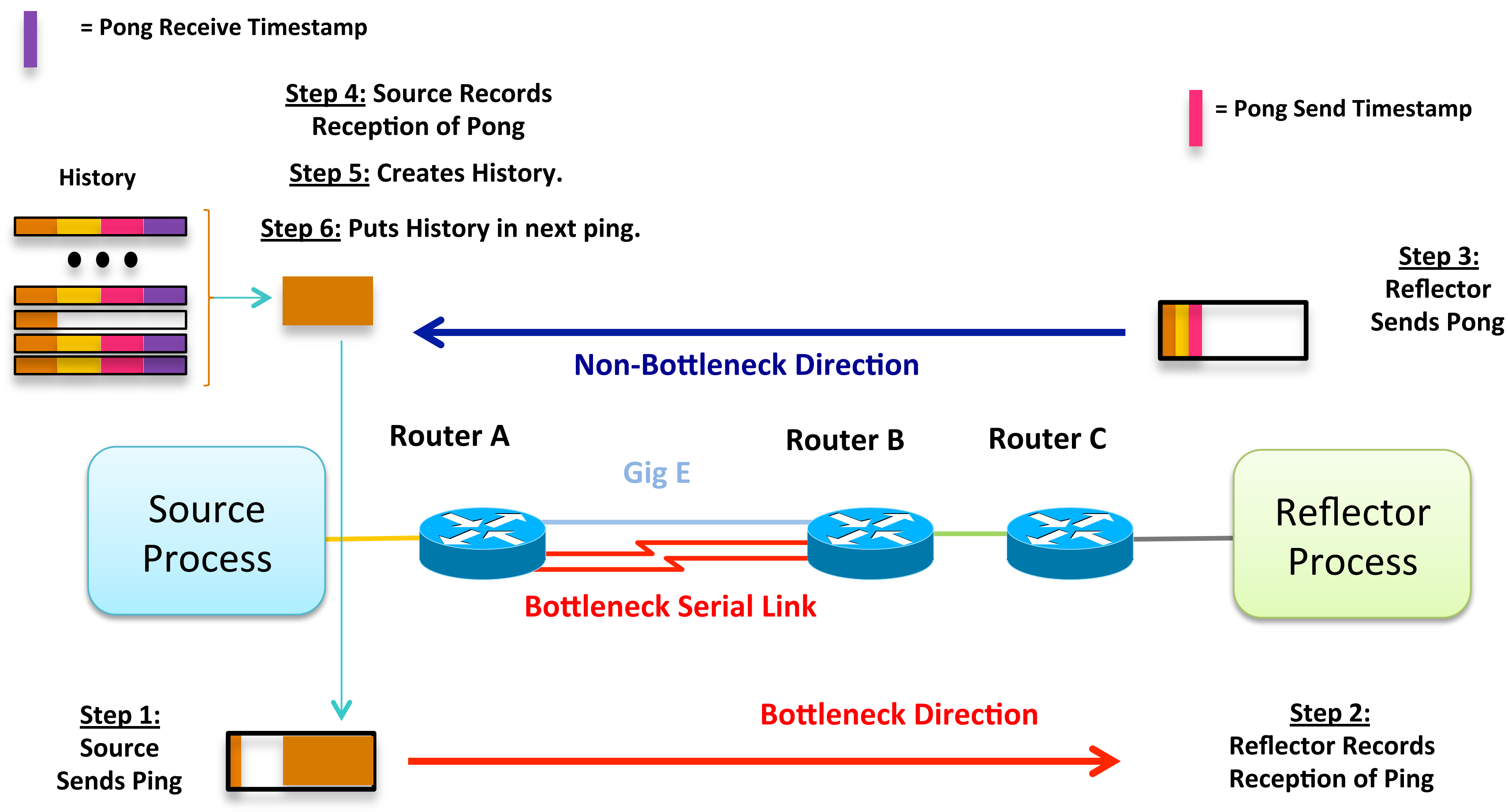


RMCAT Lab: Topology



All virtual machines are hosted on a UCS server

Delay Measurement Framework

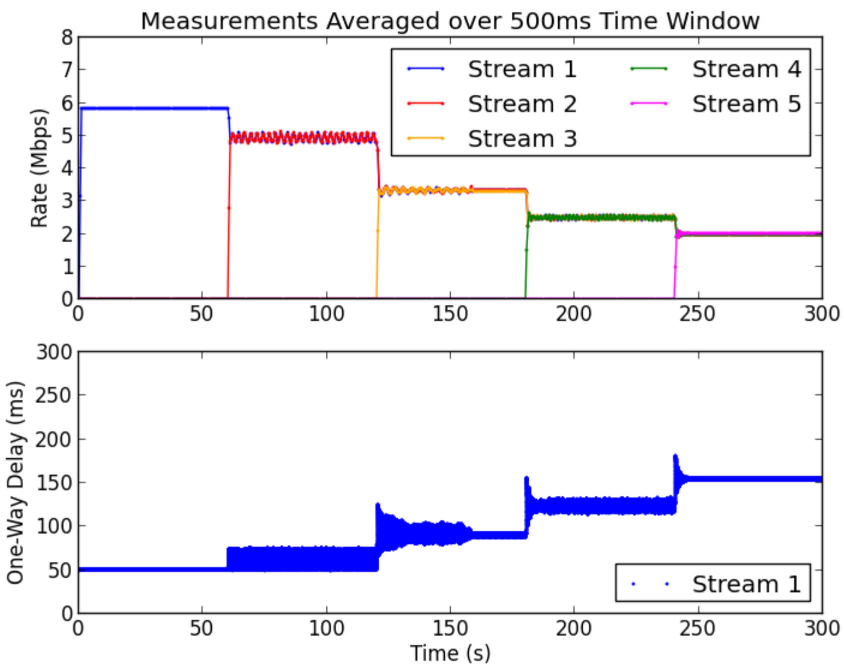


<u>Result:</u> All rates and packet lengths are known and all timestamps knowable are in packet capture!

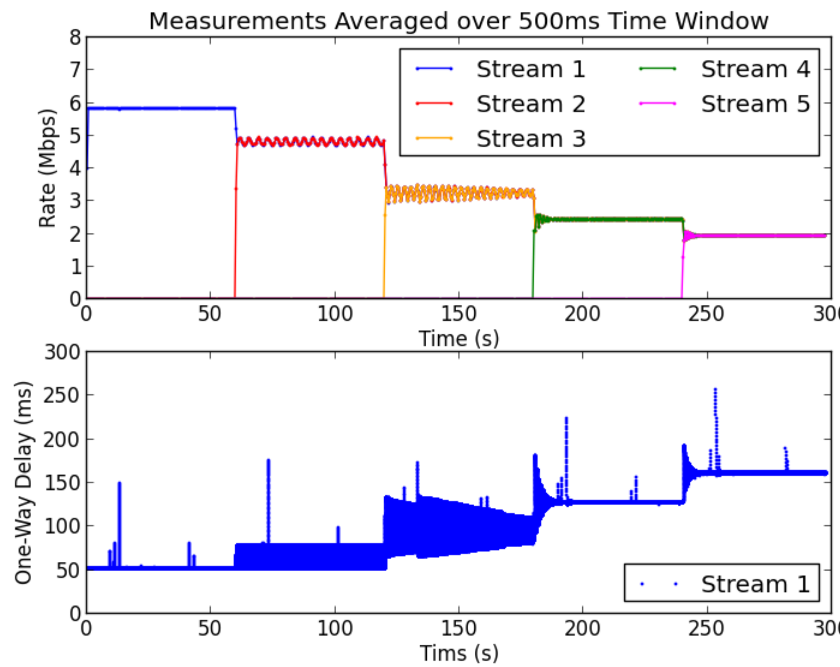
<u>NADA algorithm implemented at sender process</u>

Results from Simplified NADA Algorithm

NS-2 Simulation



Testbed



- Five NADA flows sharing a 10Mbps link
- Forward propagation delay at 50ms
- Bottleneck queue depth at 500ms
- Maximum rate at 6Mbps; minimum rate at 100Kbps

Summary of Revised NADA

- Preserves self-fairness and/or weighted bandwidth sharing
- Remains stable across a wide range of propagation delays
- Queuing at steady-state independent of path propagation
- Works with less frequent receiver feedback
- Works with relative one way delay measurements
- Robust to congestion and loss in the backward path
- Gradual linear ramp-up in response to bandwidth increase; fast non-linear reaction to bandwidth decrease

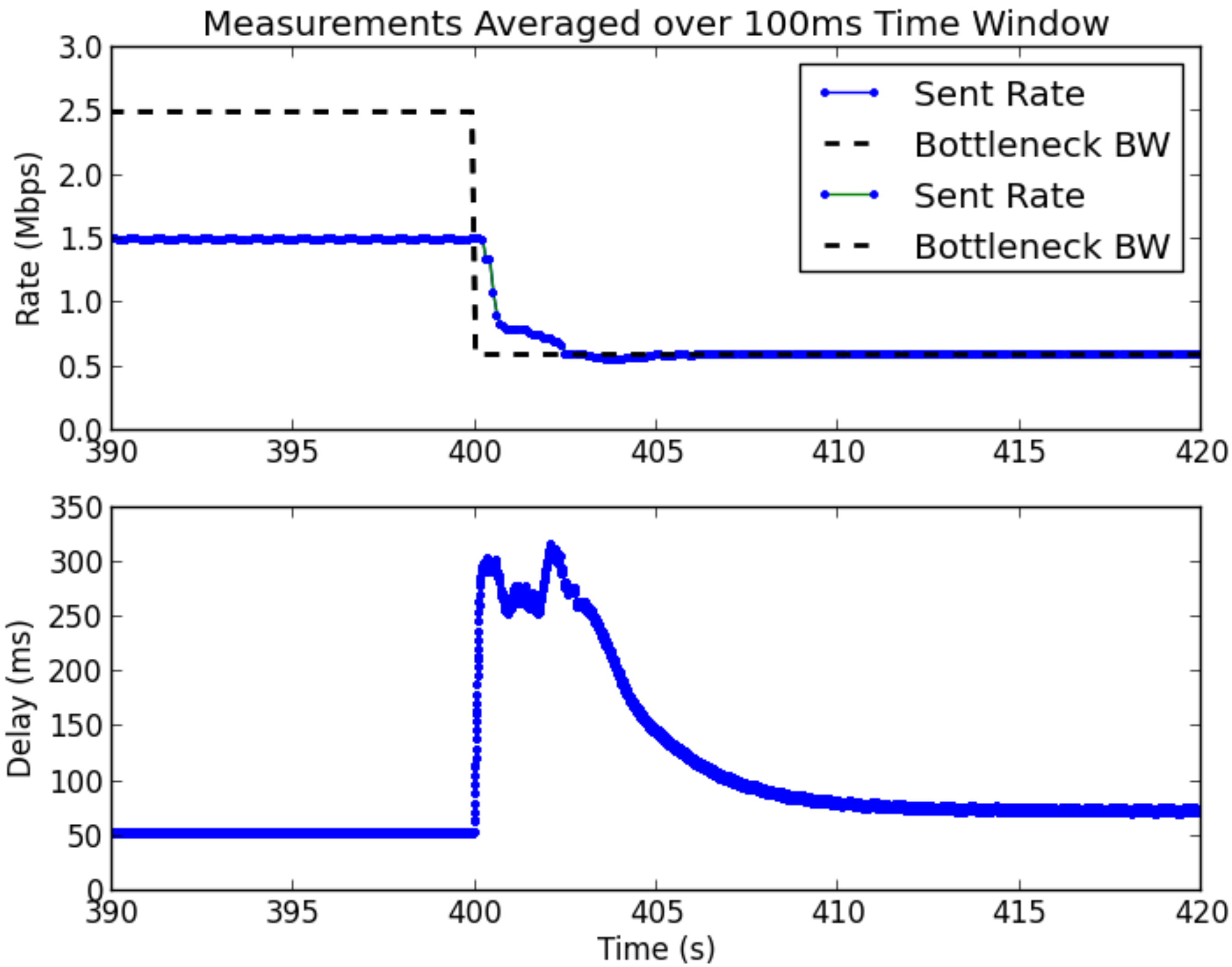
- *Open issue: need to revisit how to incorporate loss/marking information as receiver feedback*

Next Steps

- Algorithm design and analysis:
  - Incorporate loss reaction to improve NADA's behavior in the presence of competing TCP flows
  - Study the impact of algorithm parameters
- Evaluations in the RMCAT Lab:
  - Evaluation of revised NADA
  - Evaluation of other candidate algorithms

Backup Slides

Variable Available Capacity w/ Single RMCAT Flow: Zoomed-In View



Variable Available Capacity w/ Multiple RMCAT Flows: Zoomed-In View