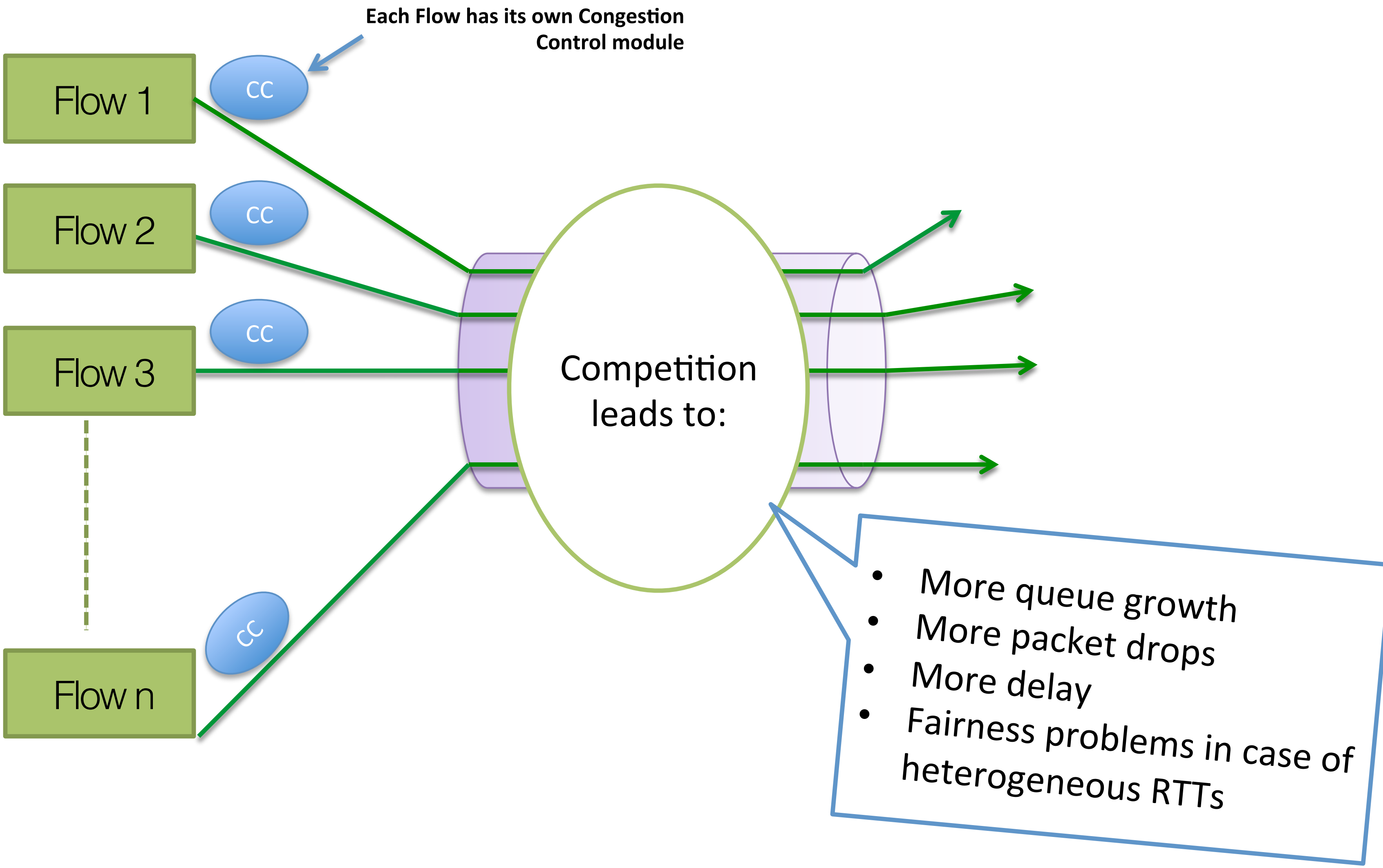
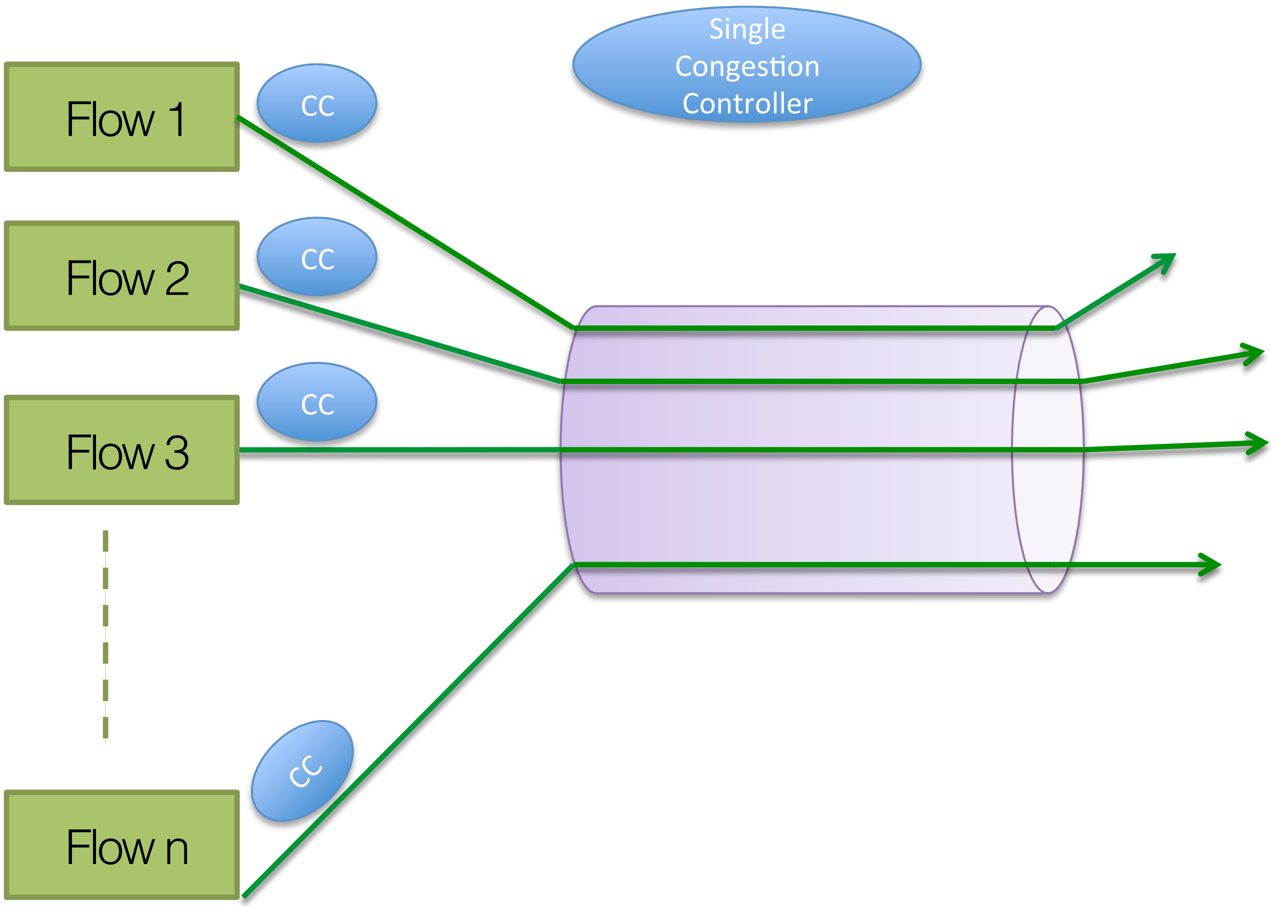
Coupled Congestion Control for RTP Media

Safiqul Islam, Michael Welzl, Stein Gjessing and Naeem Khademi
Department of Informatics
University of Oslo

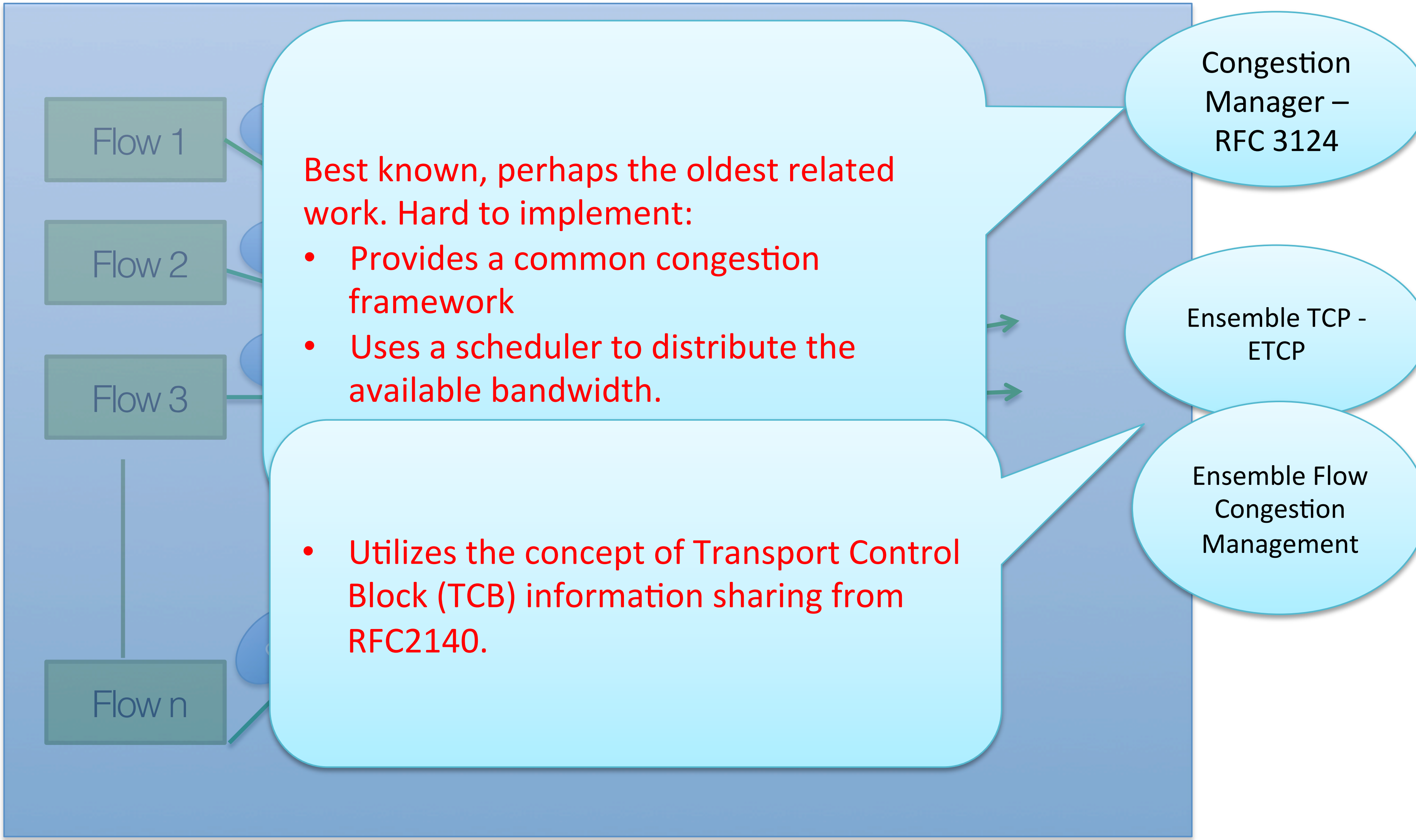
Problem statement



Problem statement – cont.



Coupled CC – related work

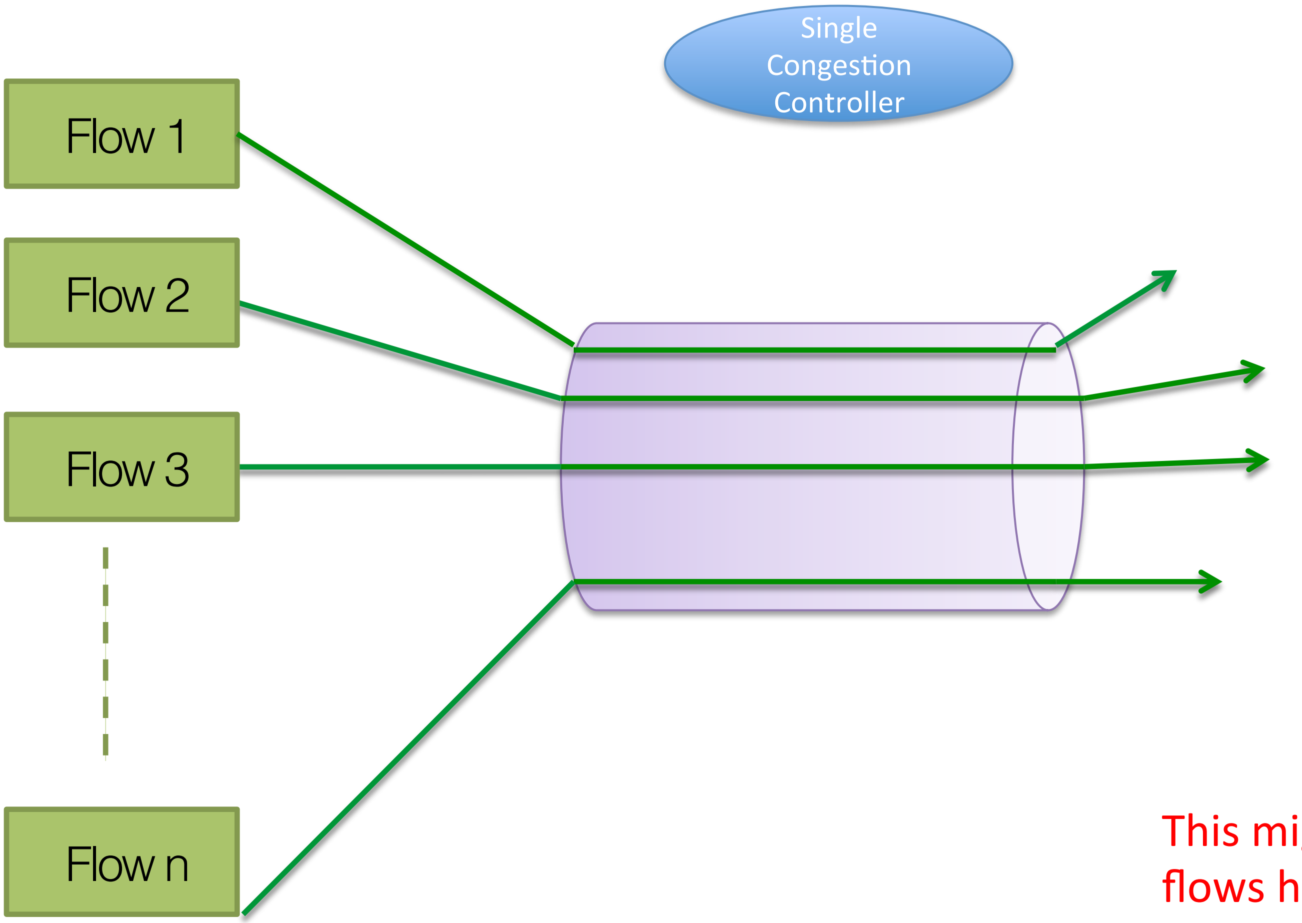


Shared bottlenecks

- Coupled CC only makes sense across a common bottleneck
  - This was ignored in prior work
  - But how to know?

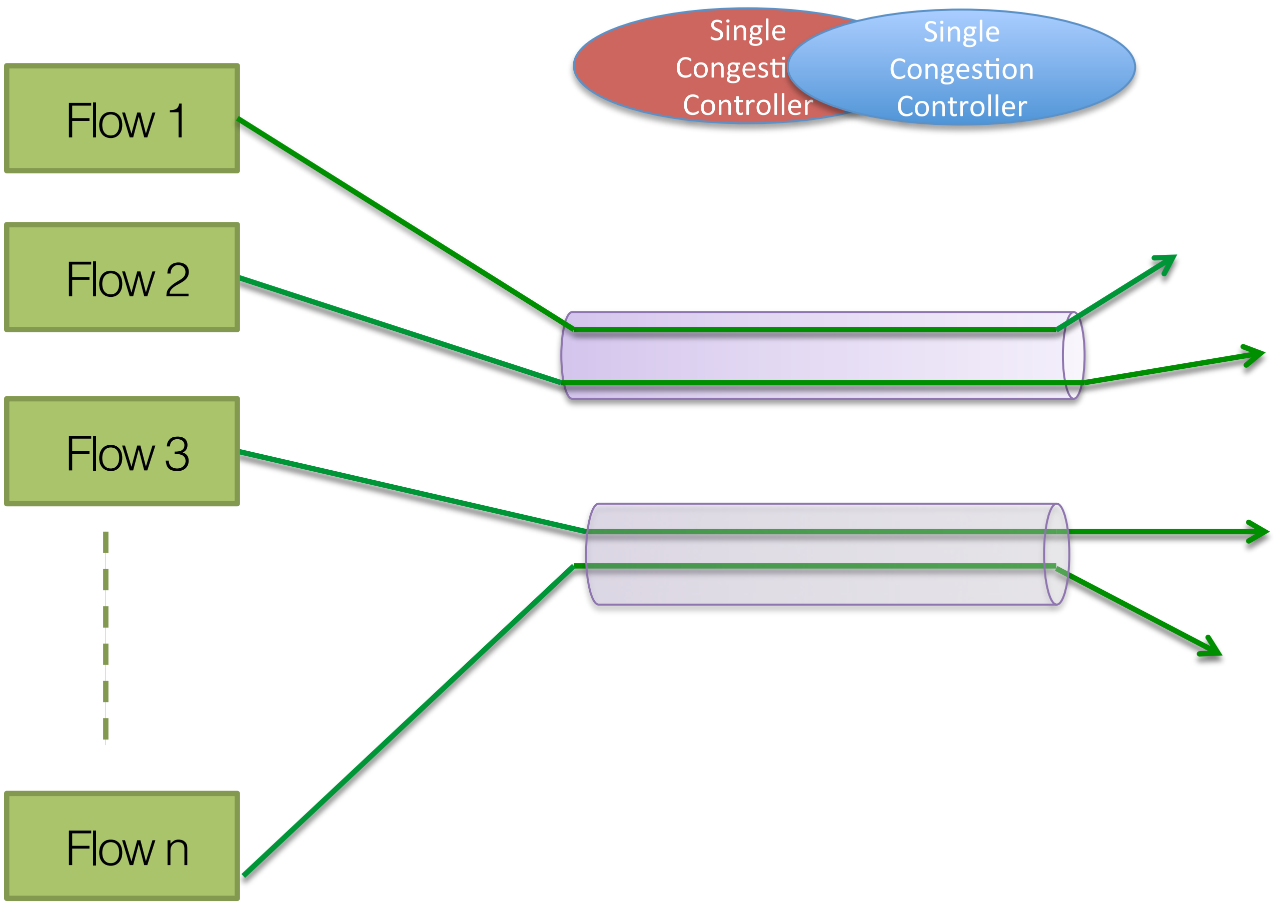
1. Multiplexing (same 5-, actually 6-tuple)
   - Fits rtcweb (coupled-cc proposed in rmcat) – but only for same source/destination hosts
2. Configuration (e.g. common wireless uplink)
3. Measurement
   - Never 100% reliable, but: <u>different receivers possible!</u>
   - Historically considered impractical, but recent work:
     David Hayes, Simone Ferlin-Oliveira, Michael Welzl: "Practical Passive Shared Bottleneck Detection Using Shape Summary Statistics", accepted for publication, IEEE LCN 2014, 8-11 September 2014

Coupled CC



This might work well for flows having same tuple but what happens when bottleneck changes?

Coupled CC

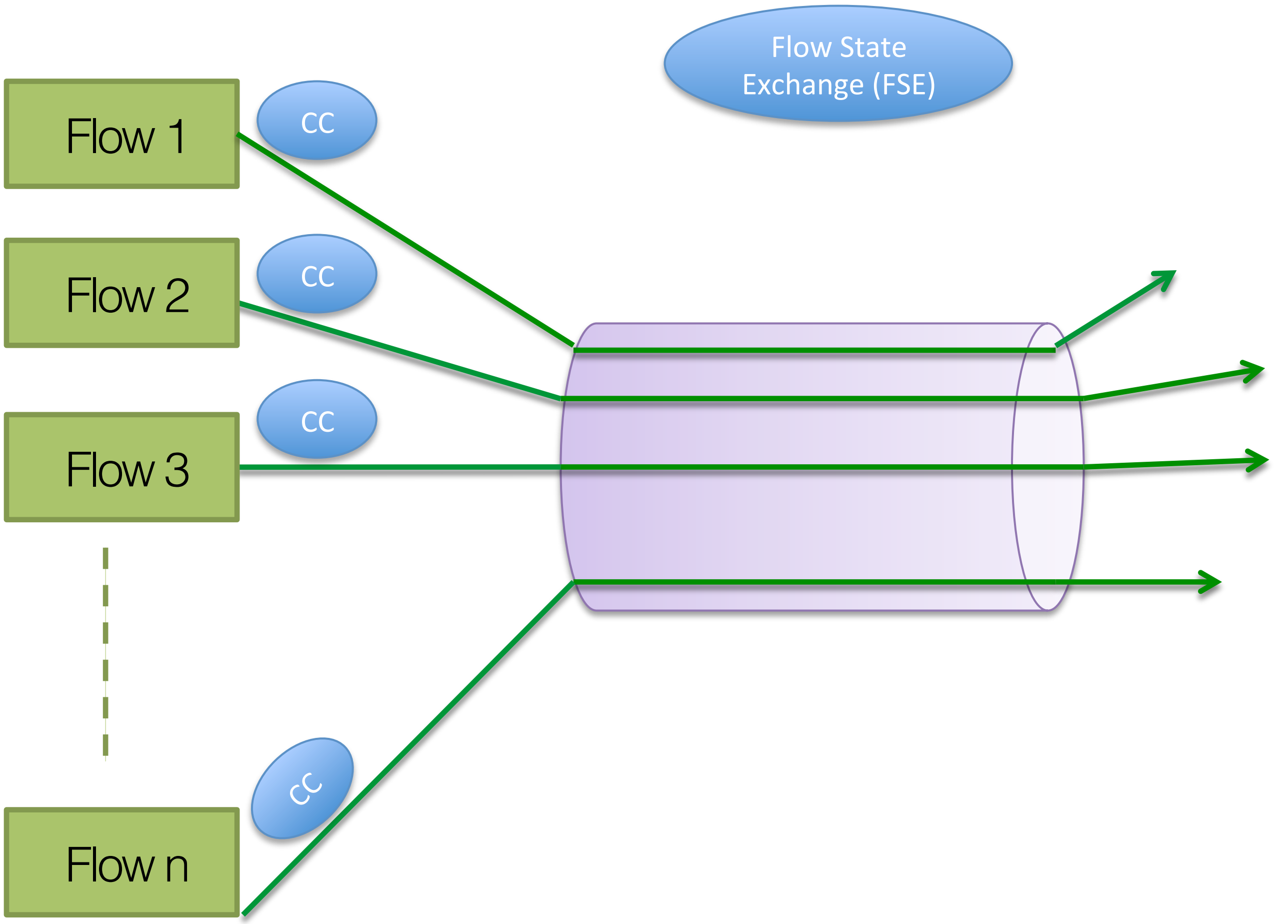


- Add another CC?
- Add another scheduler?
- What about previous CC. state?

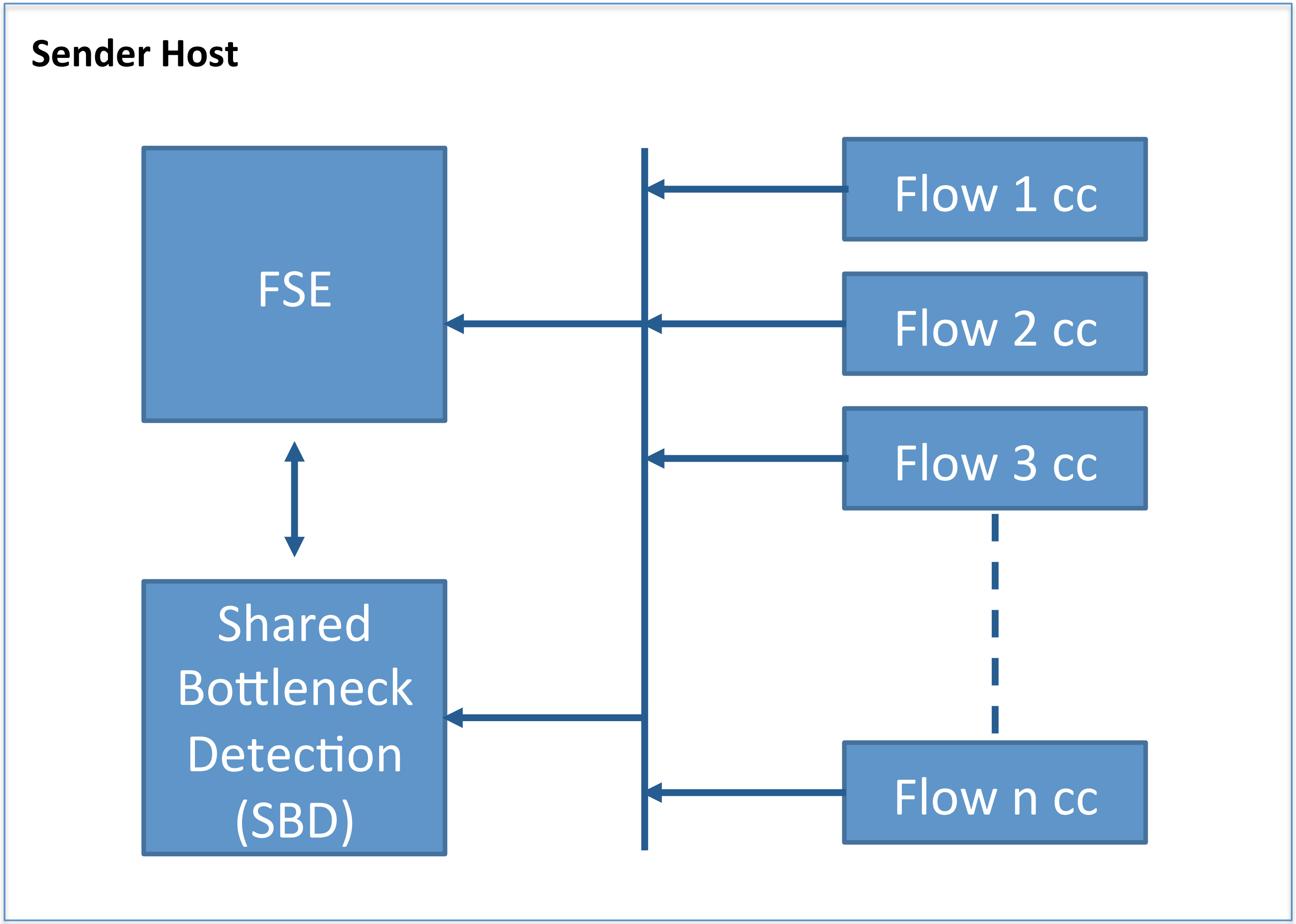
What we think:

- Better to have a simple algorithm that loosely couples existing cc with minimal changes.

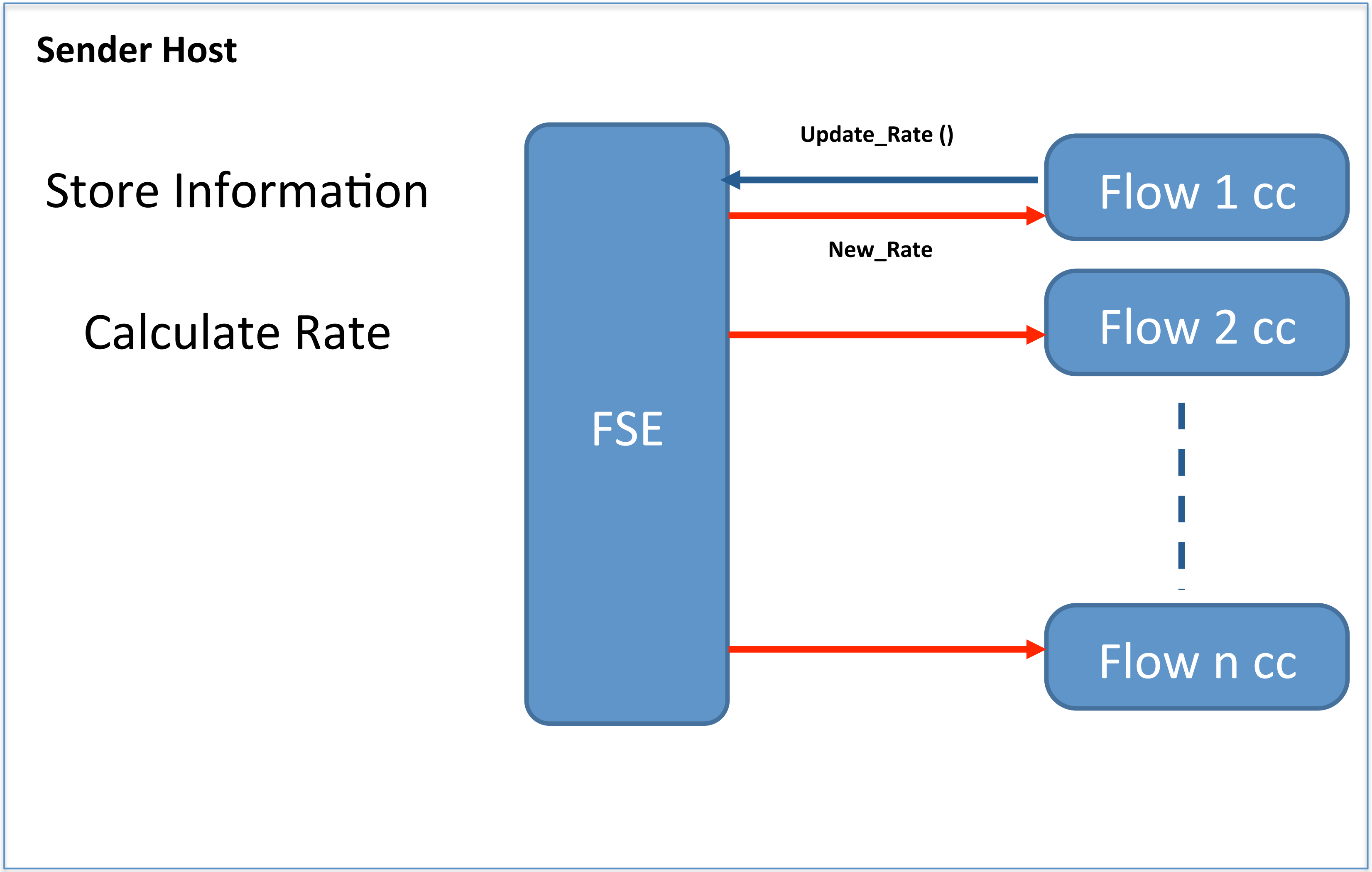
Coupled CC



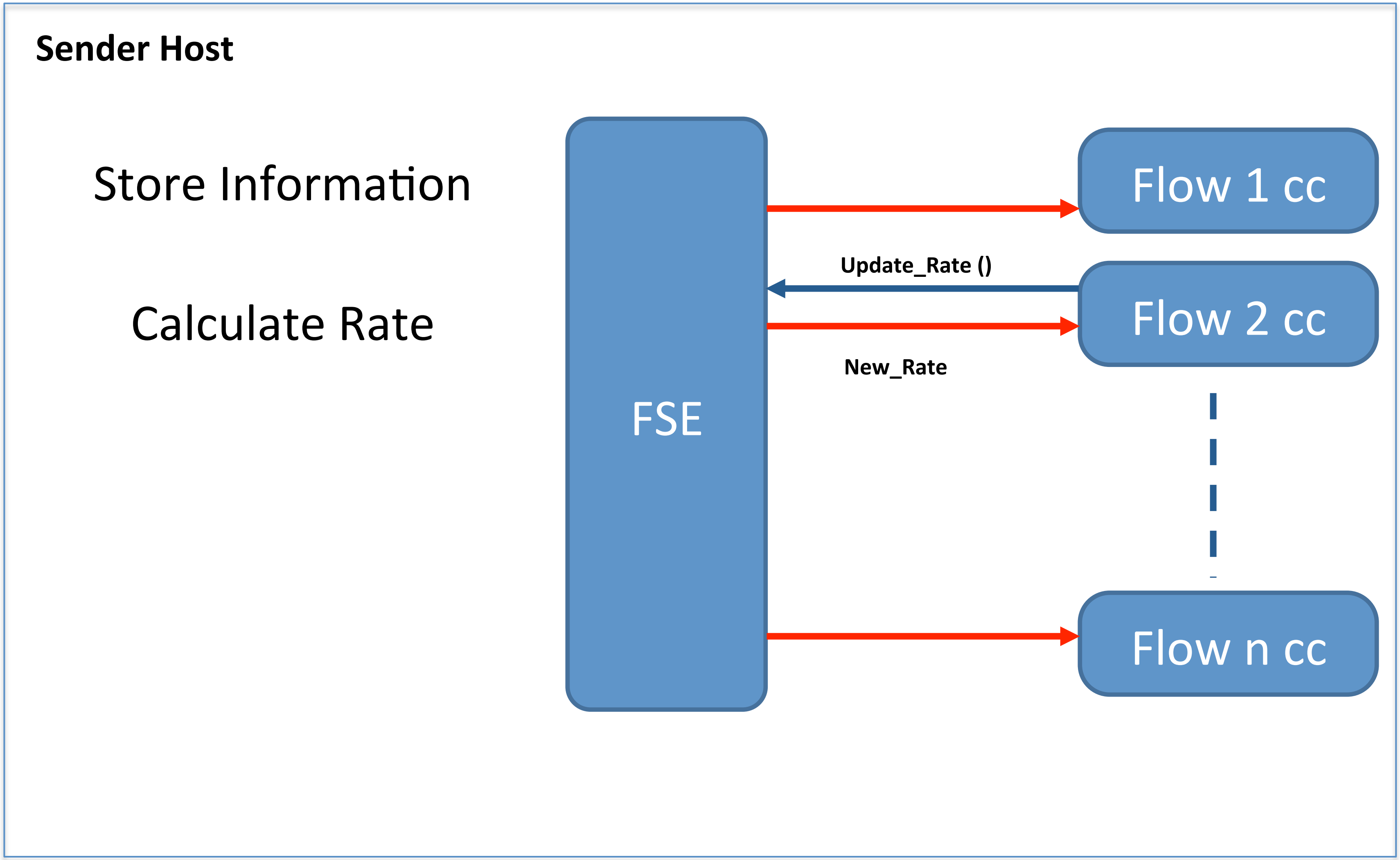
The Flow State Exchange (FSE)



The Flow State Exchange (FSE)



The Flow State Exchange (FSE)



Simple algorithm

- Every time the congestion controller of a flow determines a new sending rate, the flow calls UPDATE
  - FSE updates the sum of all rates, calculates the sending rates for all the flows and distributes them
- Results were not good
  - Details are in the paper

for all flows i in FG do

FSE_R(i) = (P(i)*ΣCR)/ΣP

send FSE_R(i) to the flow I

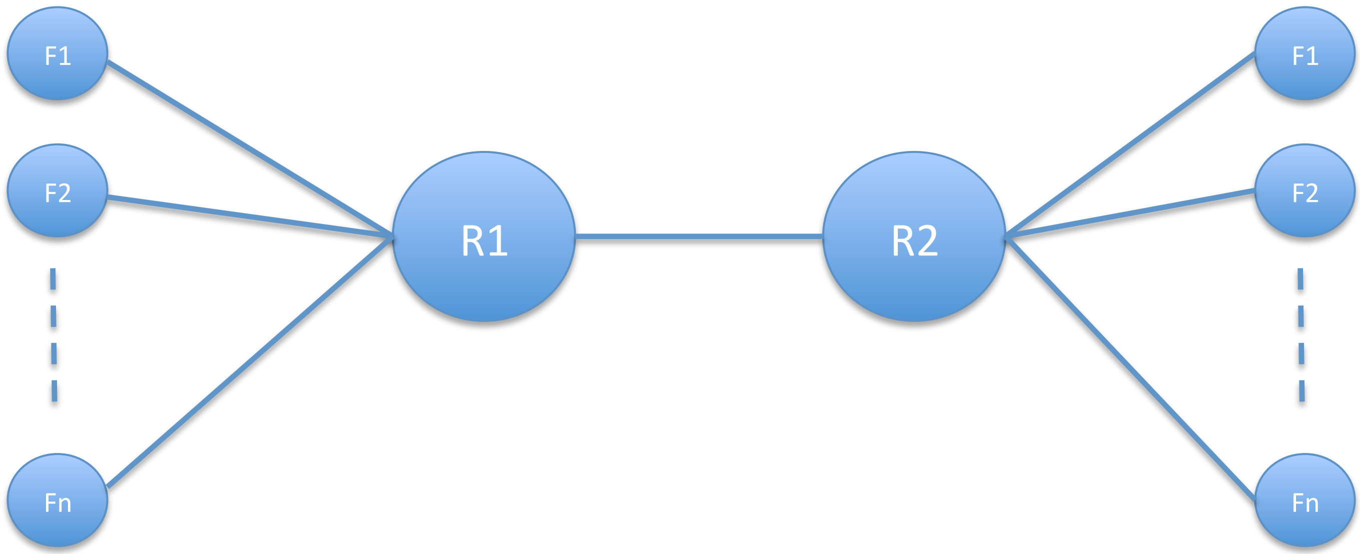
end for

Updated algorithm

Idea: reduce the rate on congestion as one flow.

- No congestion: increase the aggregate by I/N where I is the increase factor.
- Congestion: Proportionally reduce the rate to emulate the congestion response of one flow.
  - Avoid over-reacting: set a time (2RTTs) to react only once in the same loss event.

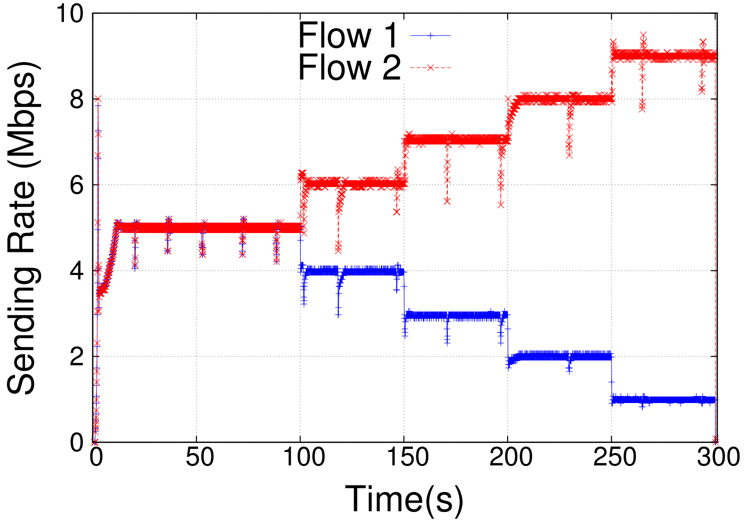
Some simulation results



- Implemented in ns-2
- Two rate based protocols:
  - Rate Adaptation Protocol (RAP)
  - TCP Friendly Transport Protocol (TFRC)
- Bottleneck – 10 Mbps, Queue-length – 62 Packets (1/2 BDP), Packet Size – 1000 Bytes, RTT – 100 ms
- All tests (except when x-axis = time) ran for 300 seconds, carried out 10 times with random start times picked from first second; stddev consistently very small (<= 0.2%)

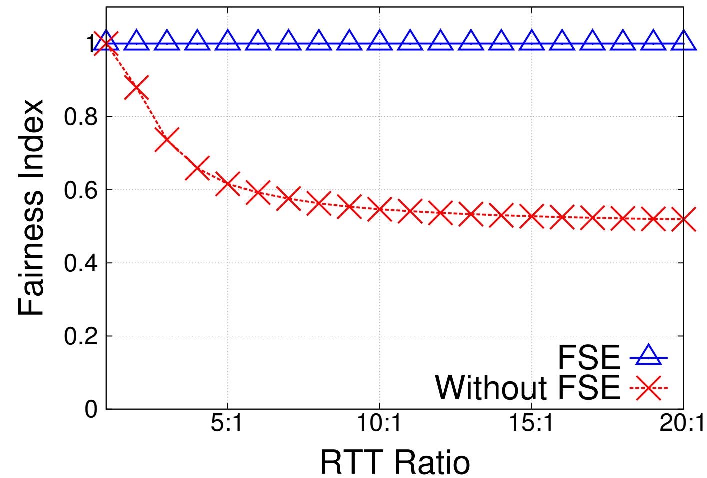
Evaluation – prioritization and fairness

Prioritization

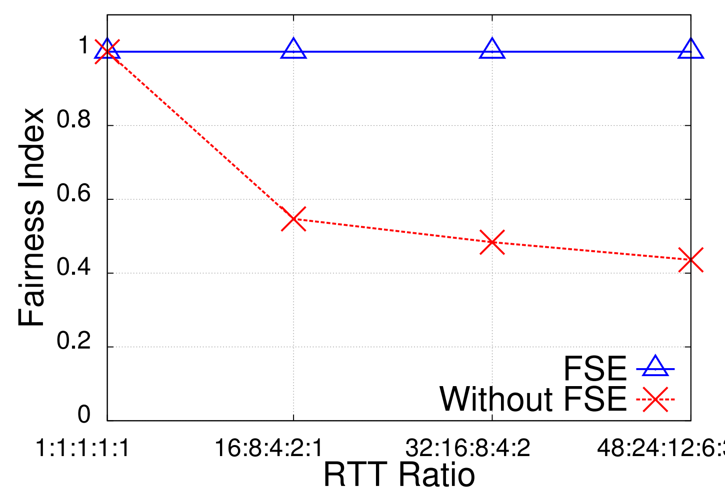


Priority of flow 1 increased over time

Fairness



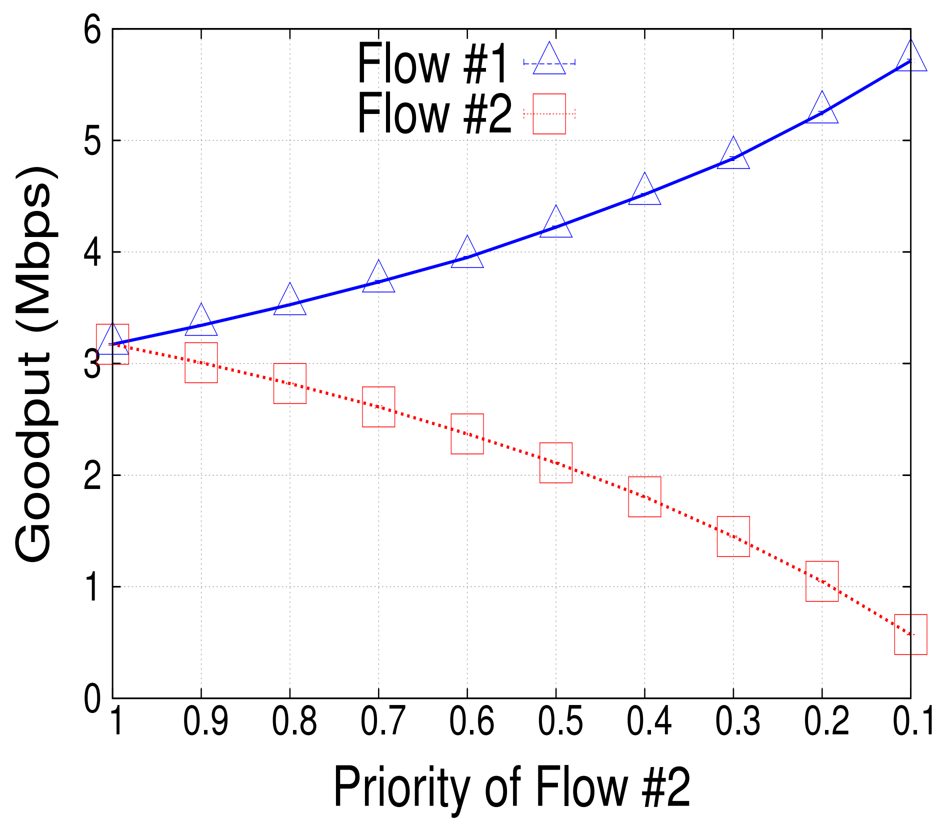
2 Flows



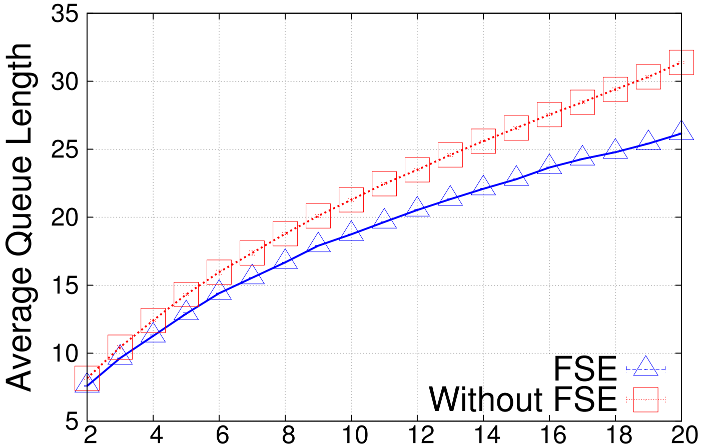
5 Flows

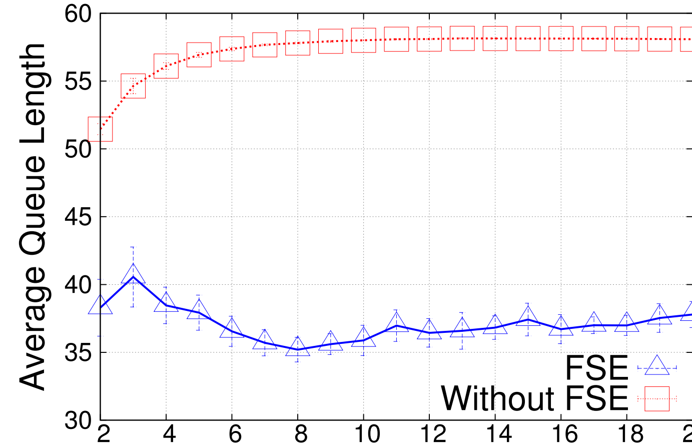
Evaluation – FSE controlled flows competing with synthetic traffic

- TMIX synthetic traffic, taken from 60 minute trace of campus traffic at the University of Carolina [TCP Evaluation suite]
  - We used the pre-processed version of this traffic which is adapted to provide an approximate load of 50%

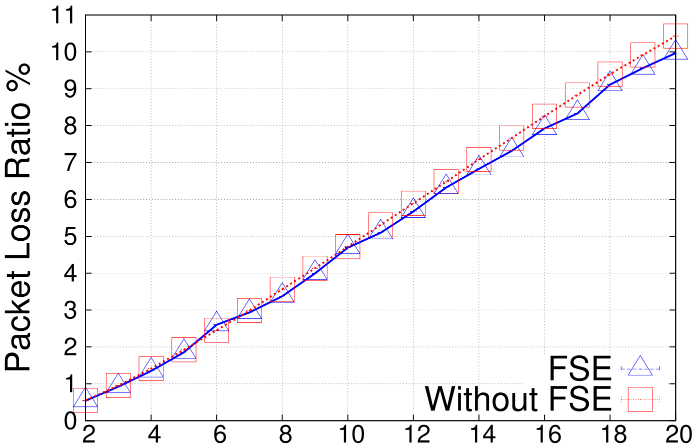


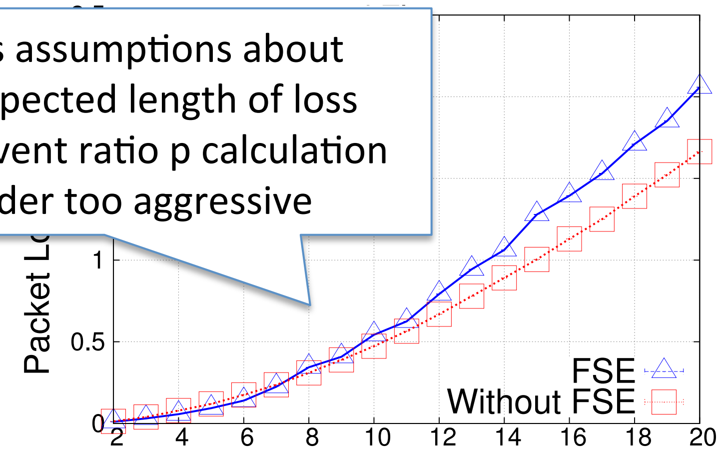
RAP **TFRC**





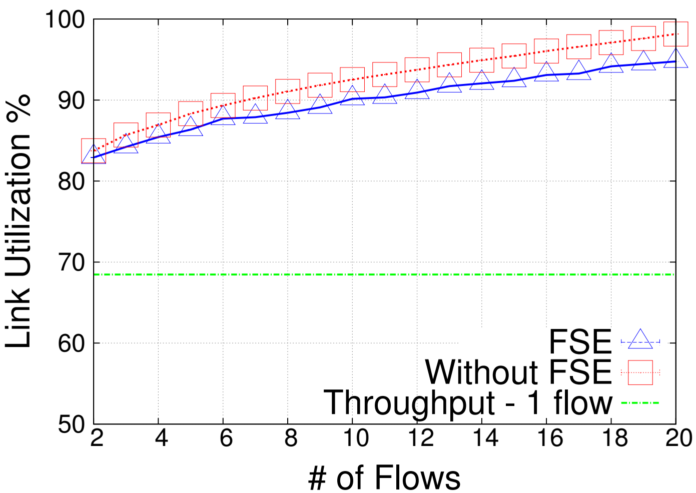
Average Queue

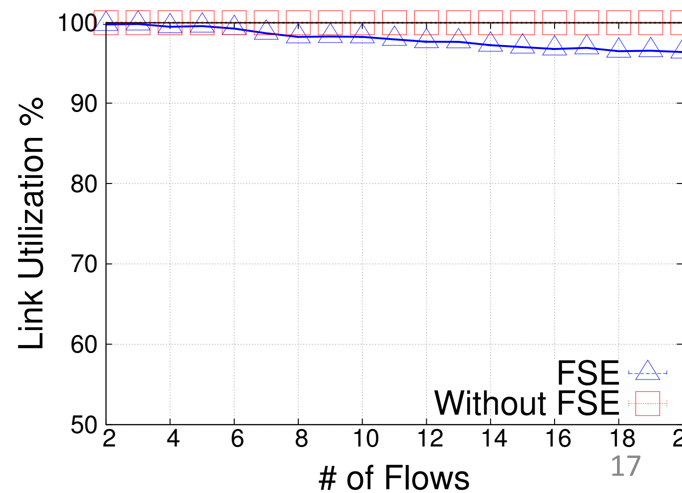




Receiver makes assumptions about sending rate (expected length of loss interval) ➔ loss event ratio p calculation wrong ➔ sender too aggressive

Packet Loss Ratio



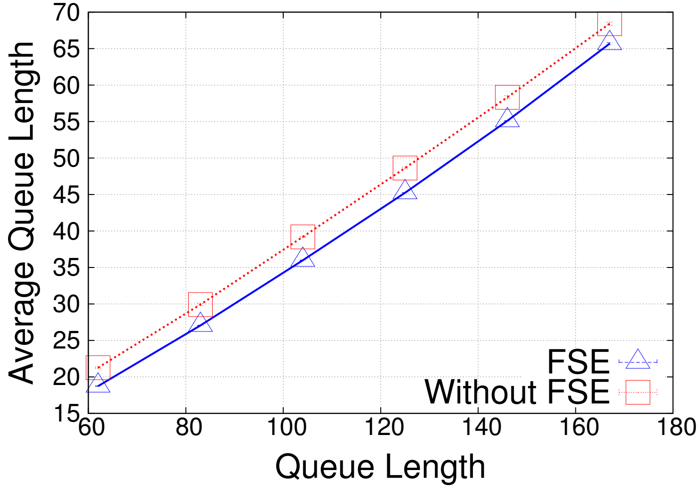


Link Utilization

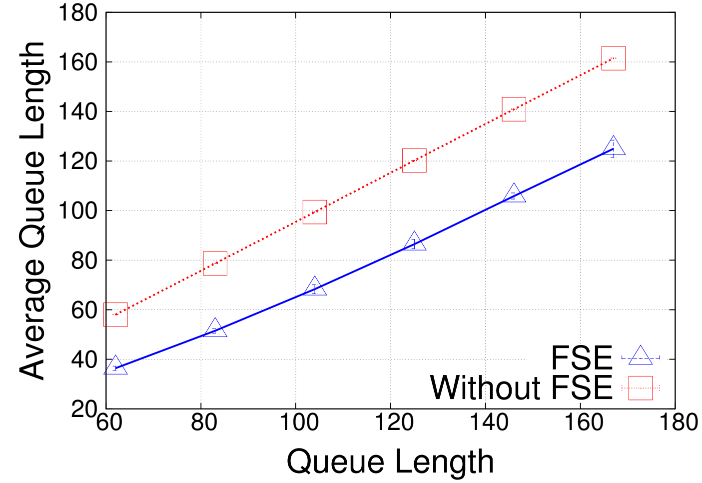
Different max Queue Lengths

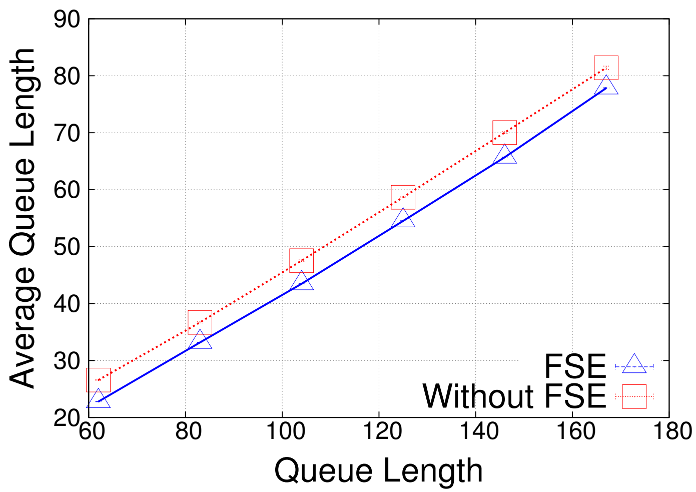
RAP

TFRC

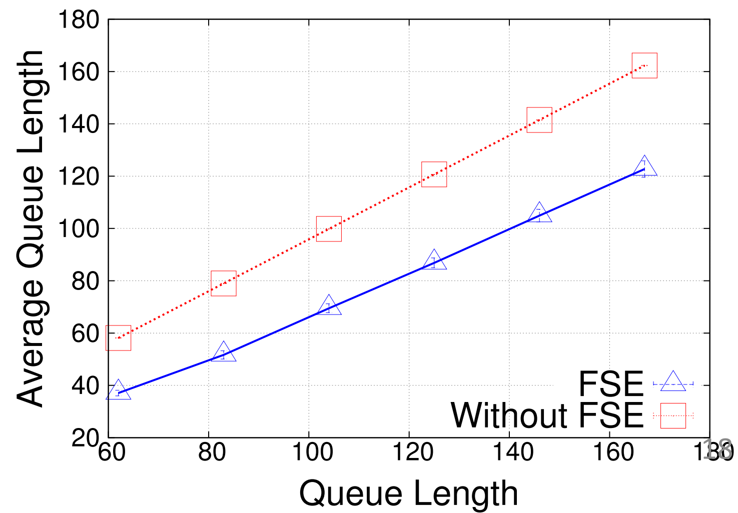


10 Flows





15 Flows

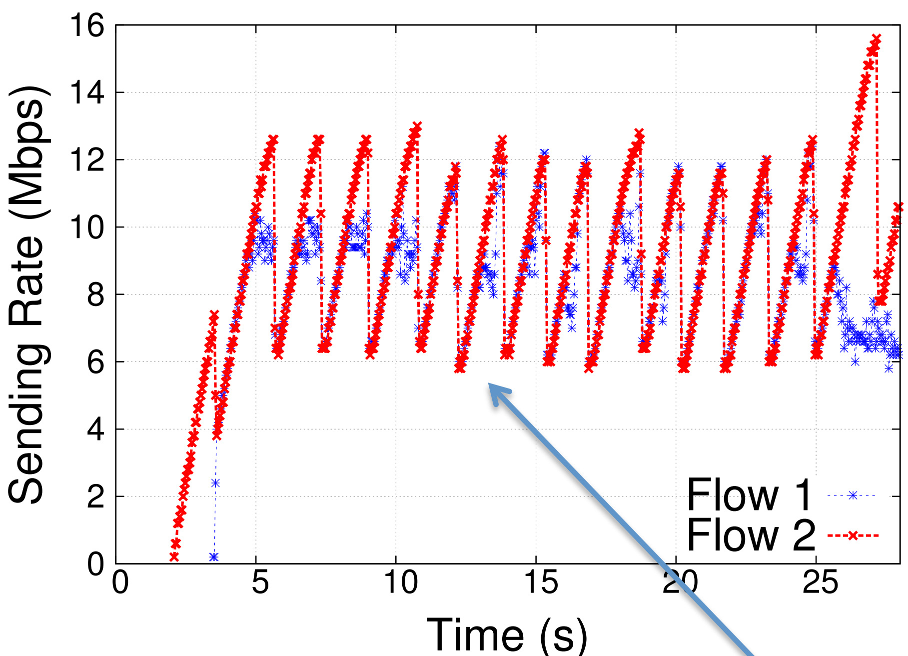


How to evaluate app-limited flows?

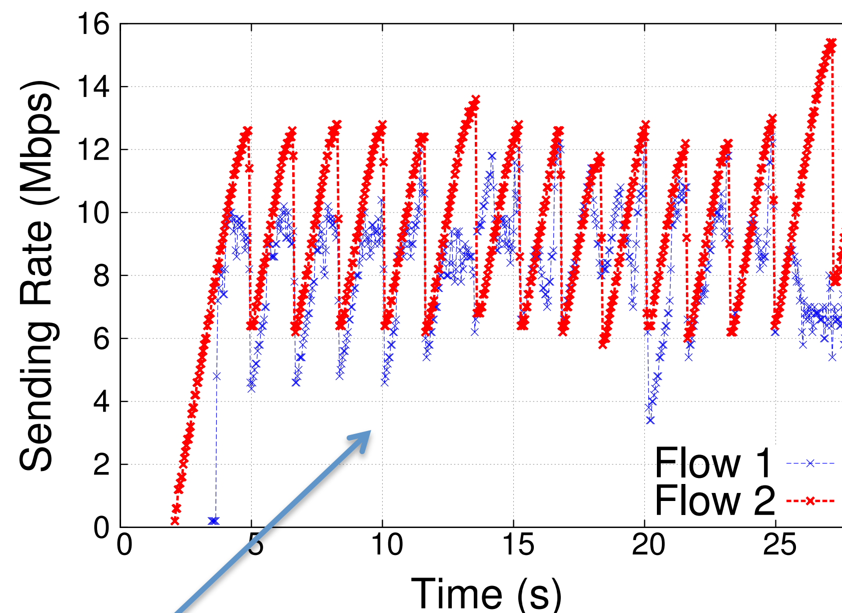
- Not easy: who is in control?
- RMCAT codec model not available yet
- From a transport point of view, the send buffer can either run empty or not, with variations in how quickly changes between these two states occur
  - We used a non-reacting video trace of a person talking in a video conference with a well-known H264 encoder (X264) to steer the app sending rate
    - I-frame in the beginning, rest was mostly P-frames

Evaluation – an application limited flow and one greedy flow (RAP)

FSE

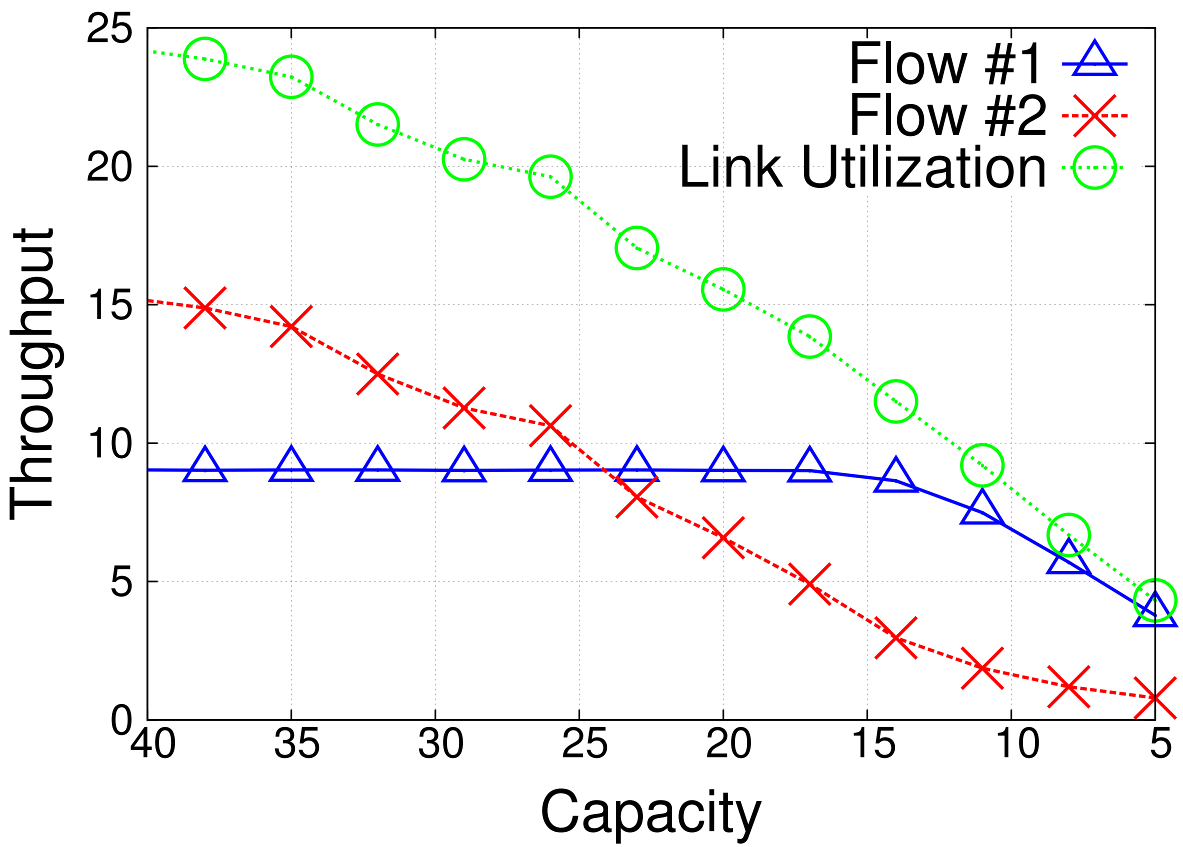


Without FSE



FSE-controlled flows proportionally reduce the rate in case of congestion; without FSE, synchronization causes app-limited flow to over-react

Using priorities to "protect" the app-limited from the greedy flow (RAP)



High-priority (1) application limited flow #1 is hardly affected by a low-priority (0.2) flow #2 as long as there is enough capacity for flow 1

Summary

- Coupled congestion control via Flow State Exchange
  - Currently proposed for WebRTC in the RMCAT group
  - Satisfies the requirements of controllable fairness with prioritization
  - Reduces queue delay and packet loss without significantly affecting throughput

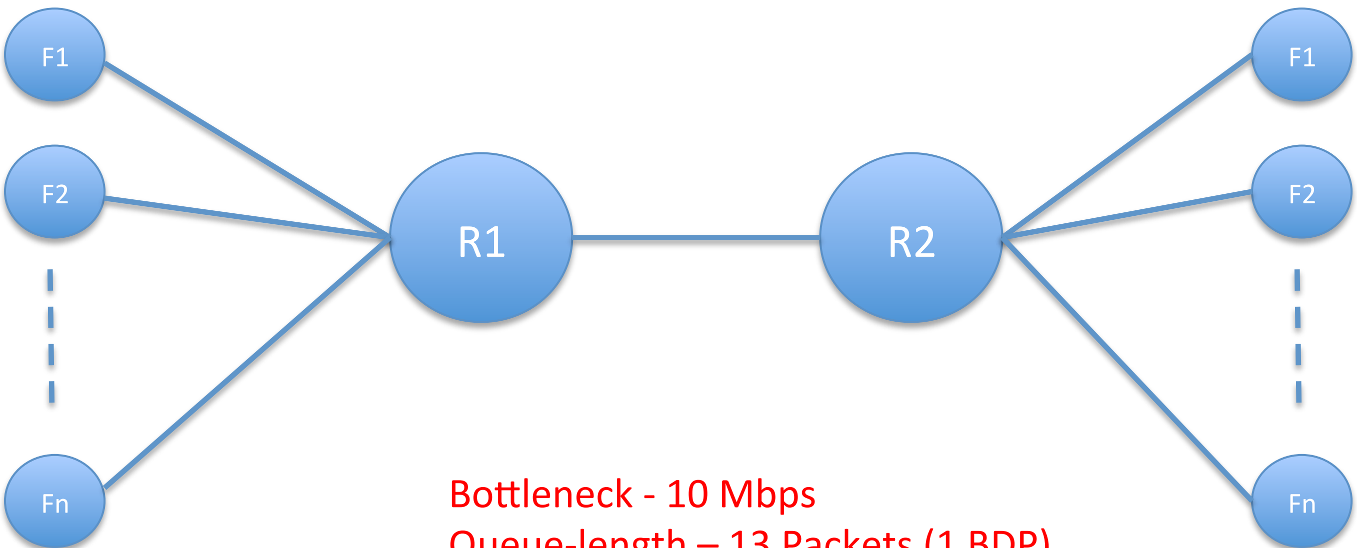
- Future work
  - Test our method in Chromium (Google's CC)
  - To incorporate WebRTC's data channel, we will investigate coupling with window-based protocol too

That’s all !!

Questions?

Backup slides

Experimental setup (simple algorithm)

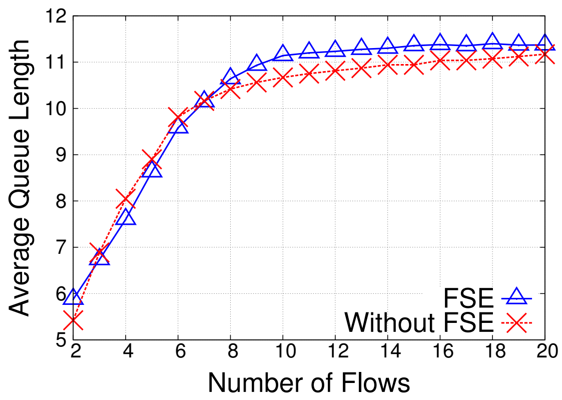


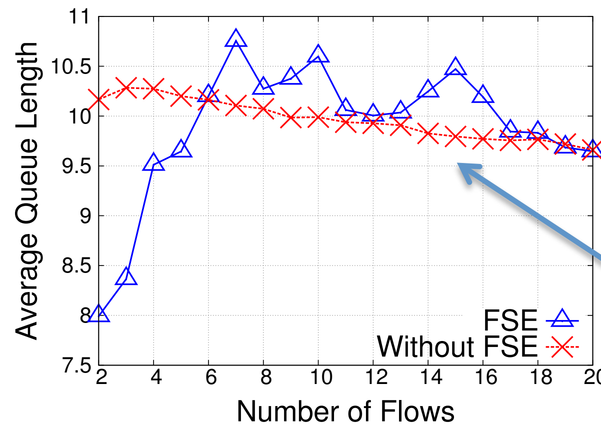
Bottleneck - 10 Mbps
Queue-length – 13 Packets (1 BDP)
Packet Size – 1000 Bytes
Senders have always data to send

Results – simple algorithm

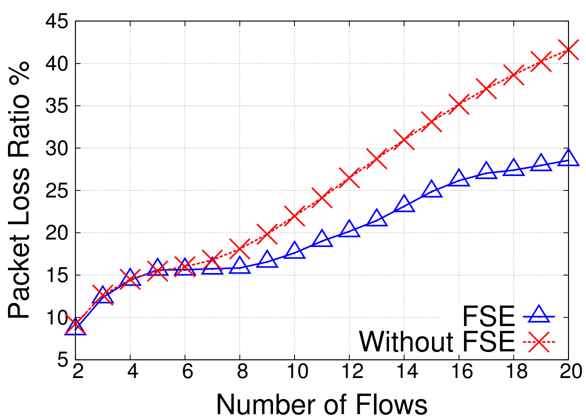
RAP

TFRC

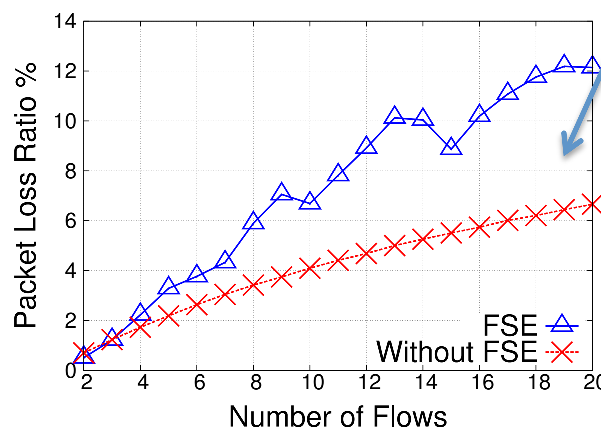




Average Queue Length

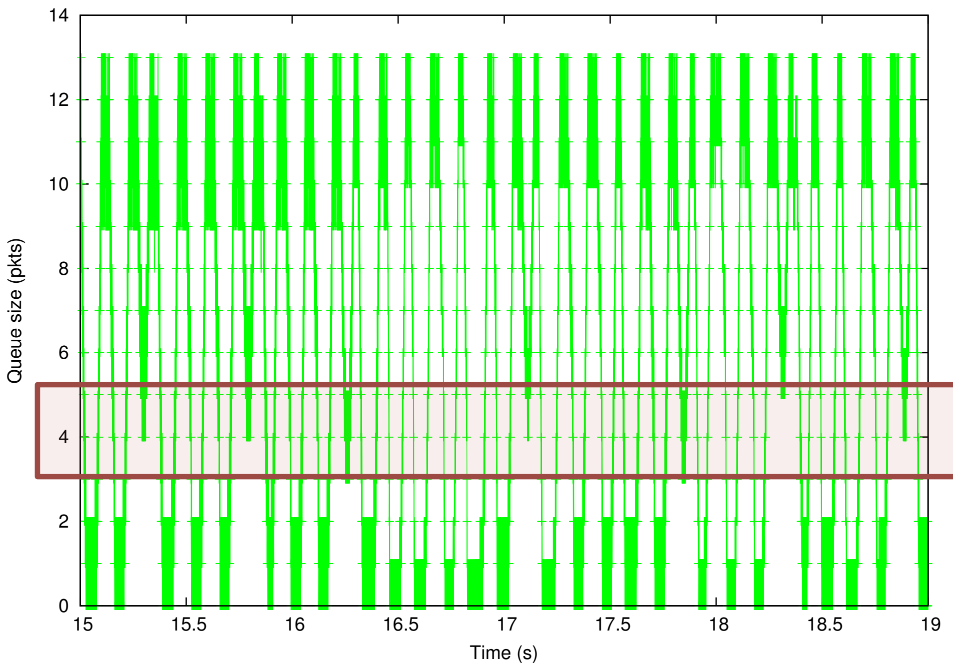


Packet Loss Ratio

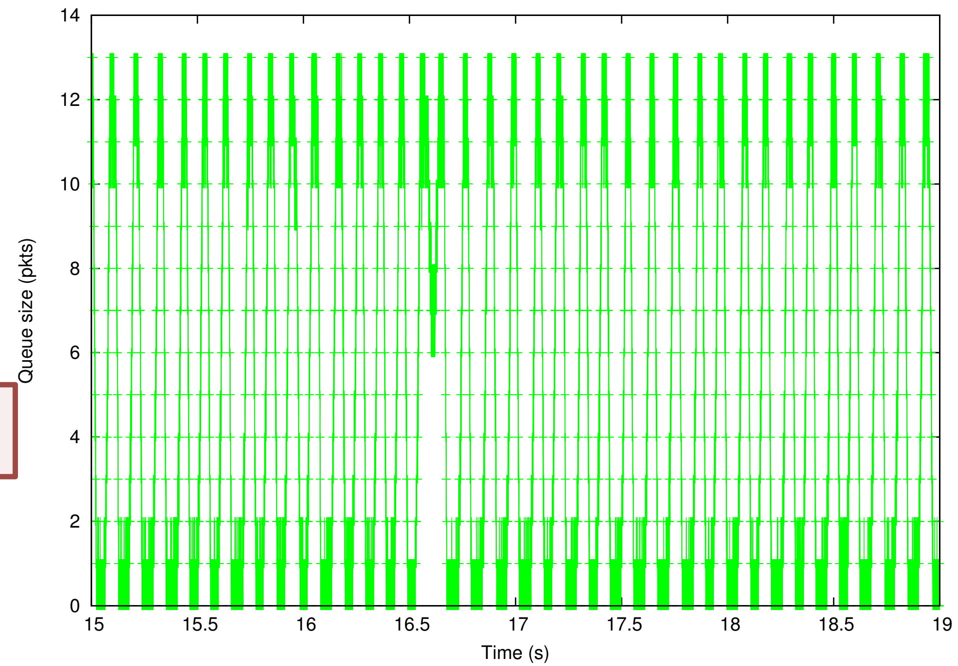


Why?

What's going on? (simple algorithm)



With FSE



Without FSE

- Queue drains more often without FSE
  - Should emulate the congestion response of one flow
    - FSE: 2 flows with rate X each; one flow halves its rate: 2X ➔ 1 ½X
    - When flows synchronize, both halve their rate on congestion, which halves the aggregate rate
    - We want that ! 2X ➔ 1X