neat

Lightweight and Flexible Single-Path Congestion Control Coupling

Safiqul Islam

Networks and Distributed Systems Group
Department of Informatics
University of Oslo

Supervisors

Michael Welzl
Univ of Oslo

Stein Gjessing
Univ of Oslo

Dissertation Committee

Joerg Ott
TU Munich

Colin Perkins
Univ of Glasgow

Tor Skeie
Simula Research Laboratory

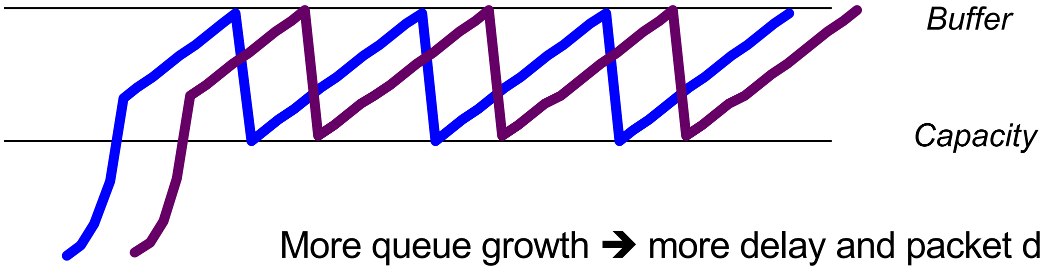
NOMS'18 Conference
April 26, 2018

Motivation

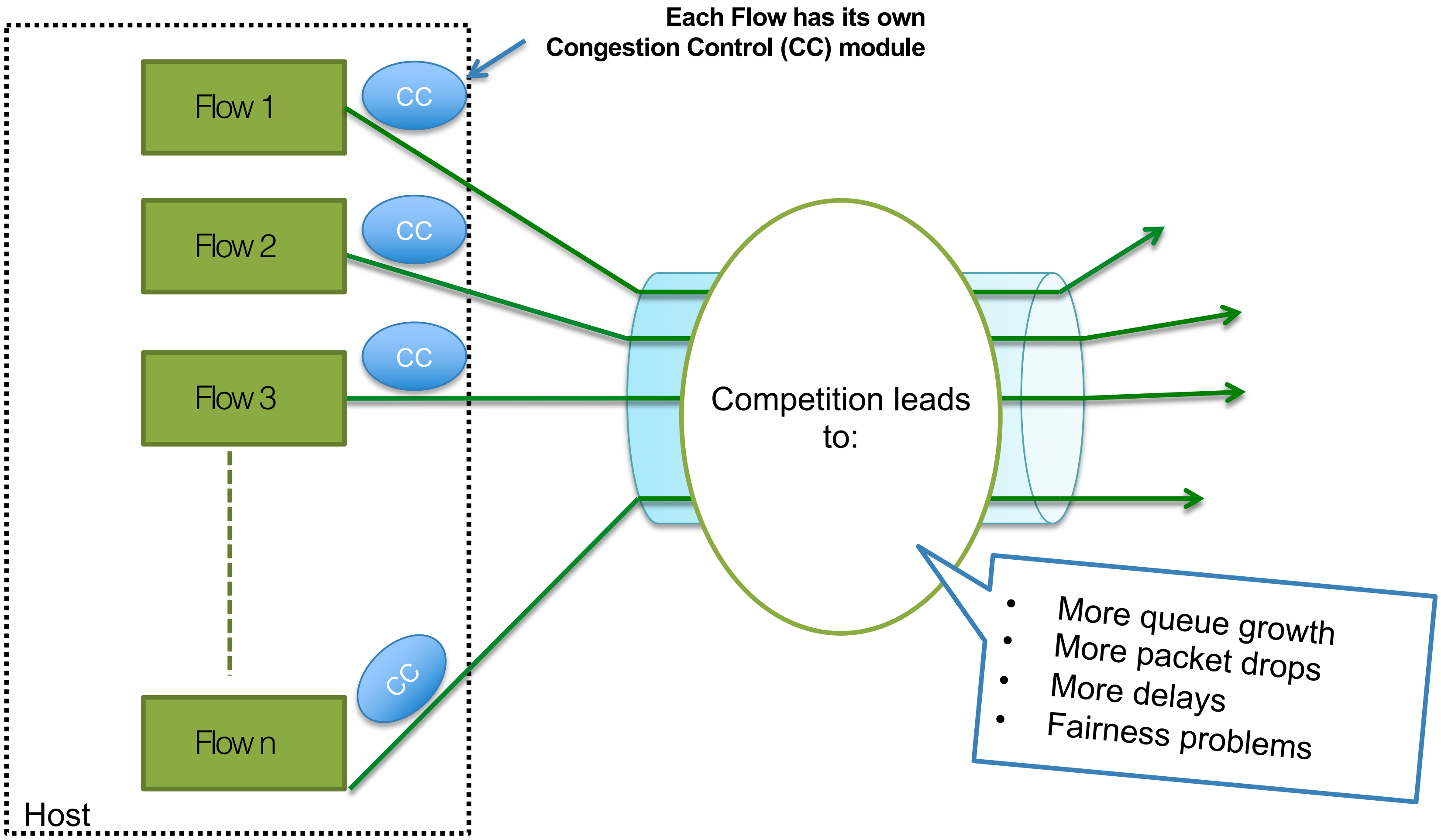
Problem Statement

Problem Statement

- Congestion control probes for the available capacity to reach a certain notion of fairness

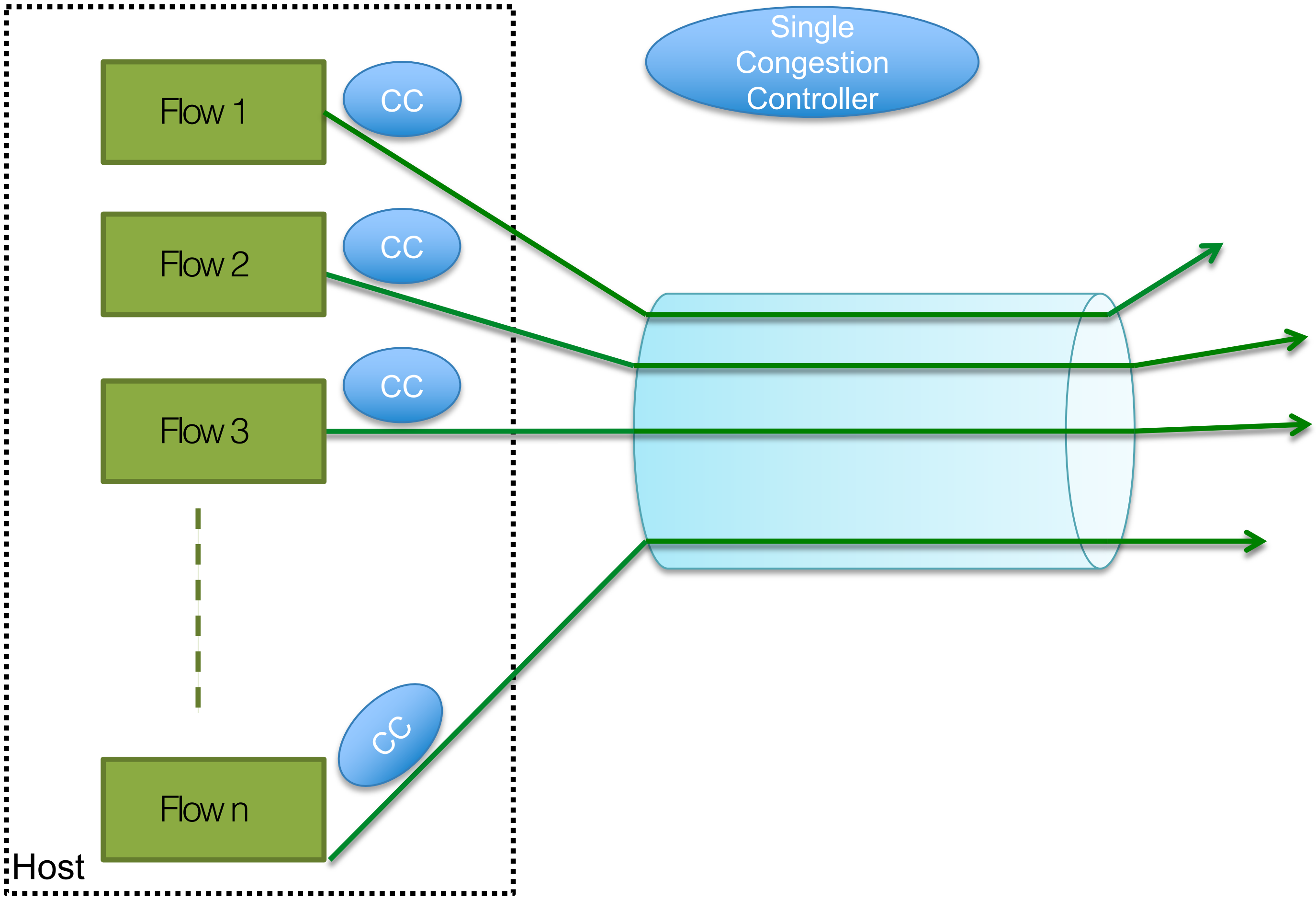


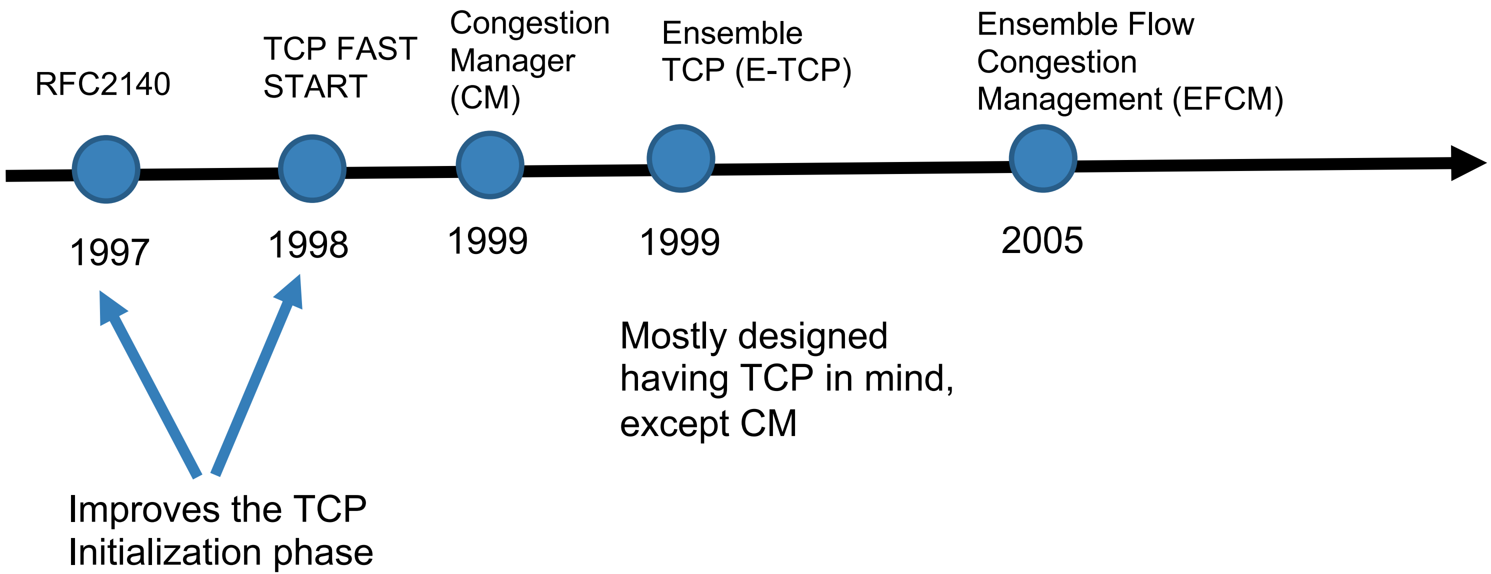
More queue growth ➔ more delay and packet drops

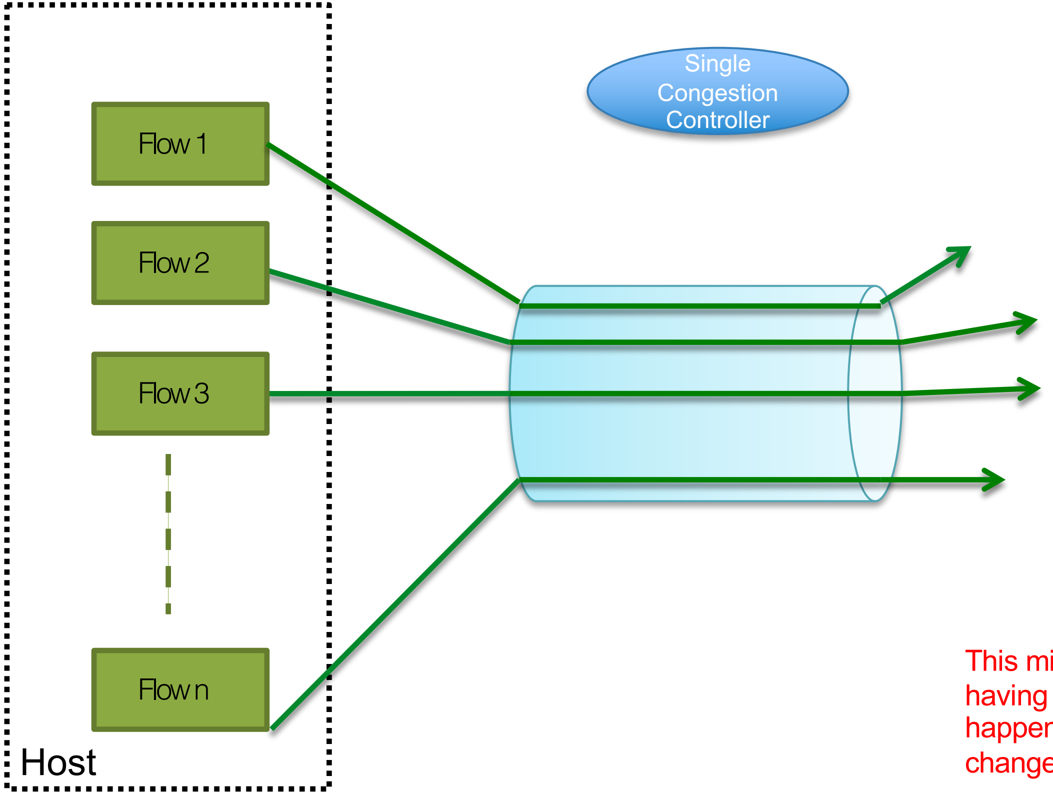

Each Flow has its own
Congestion Control (CC) module
CC
Flow 1
CC
Flow 2
CC
Flow 3
CC
Flow n
Host
Competition leads
to:
• More queue growth
• More packet drops
• More delays
• Fairness problems

Would it be possible to solve these problems?

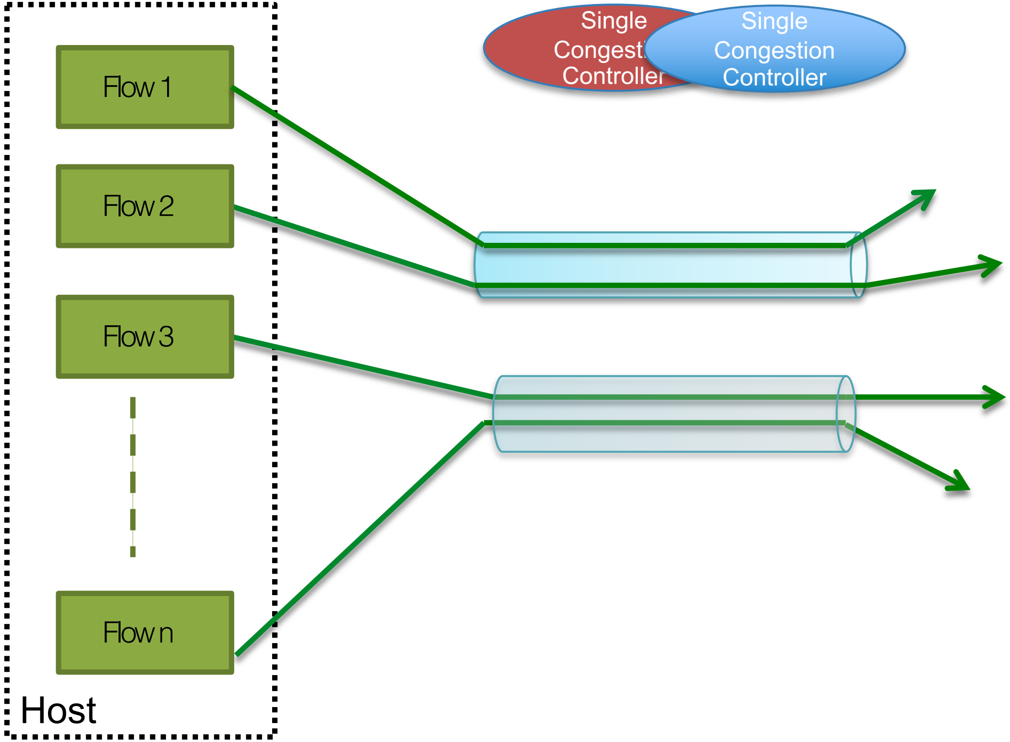
YES


Single Congestion Controller
Flow 1
CC
Flow 2
CC
Flow 3
CC
CC
Flow n
Host


RFC2140
TCP FAST START
Congestion Manager (CM)
Ensemble TCP (E-TCP)
Ensemble Flow Congestion Management (EFCM)
1997
1998
1999
1999
2005
Mostly designed having TCP in mind, except CM
Improves the TCP Initialization phase



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



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

#1

Simple and Flexible

#2

Ensure a common bottleneck

#3

Make it Generic --- Apply to different CCs

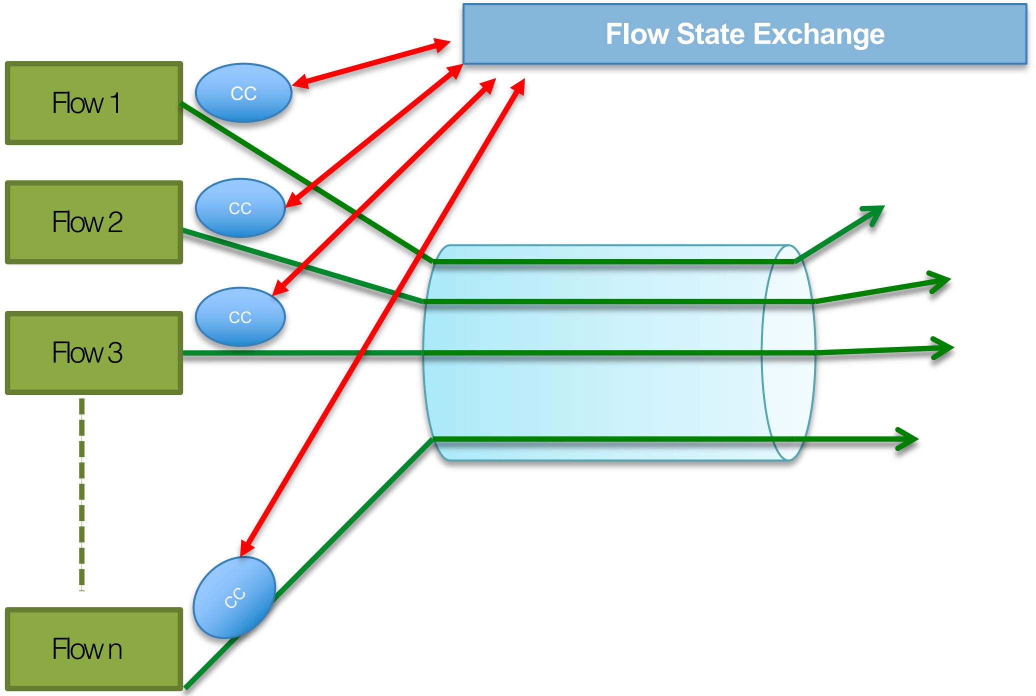
#4

Reduce overall delay and losses

#1

Simple and Flexible

Flow State Exchange (FSE)



I

Simple and Flexible

Two Variants

Passive

Maintains the state of the ensemble and makes it available to **only** the flow requesting a new rate.

- Less signaling
- Minimal Changes
- Homogeneous RTTs

Active

Actively initiates communication with all the flows.

#2

Ensure a common bottleneck

Shared Bottlenecks

- Managing flows with a common FSE: only across a common bottleneck
  - This was ignored in prior work (CM, E-TCP, EFCM)
  - But how to know?

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

#3 Apply to different CCs

Simple → Sophisticated

Loss-based	RAP	TFRC	TCP
Delay-based	LEDBAT	NADA	GCC

#4

Reduce overall loss and delay

Evaluations

Coupled Congestion Control for RTP Media

- **RAP and TFRC** *SIGCOMM CCR'14, CSWS'14*
- **WebRTC congestion controllers (NADA and GCC)** *NOMS'16*
- **I-D – IETF RMCAT WG** *RFC editor Queue*

TCP congestion control coupling

- **Novel ACK clock mechanism for initializing IW** *ANRW'16*
- **RFC 2140bis** *TCPM WG Draf*
- **TCP-IN-UDP encapsulation** *IRTF ICCRG Draft*
- **TCP CCC** *GI'18*
- **Combining different CCC (LEDBAT and TCP)** *IMCEC'16*

Evaluations

Coupled Congestion Control for RTP Media

- RAP and TFRC
- WebRTC congestion controllers (NADA and GCC)
- I-D – IETF RMCAT WG

SIGCOMM CCR’14, CSWS’14

NOMS’16

RFC editor Queue

RAP and TFRC

- 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

```
for all flows i in FG do
    FSE_R(i) = (P(i)*ΣCR)/ΣP
    send FSE_R(i) to the flow I
end for
```

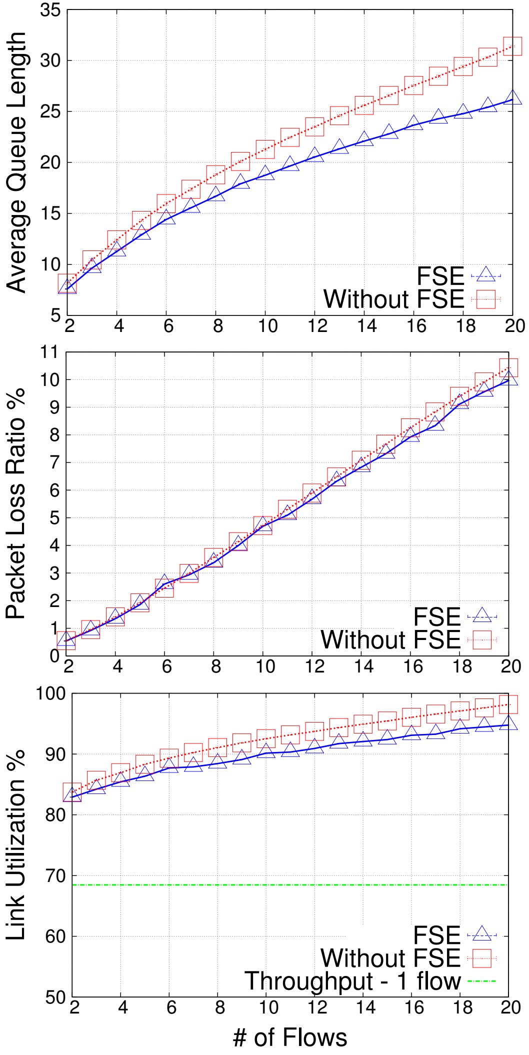
- Results were not good
  - Details are in the paper

RAP and TFRC

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.

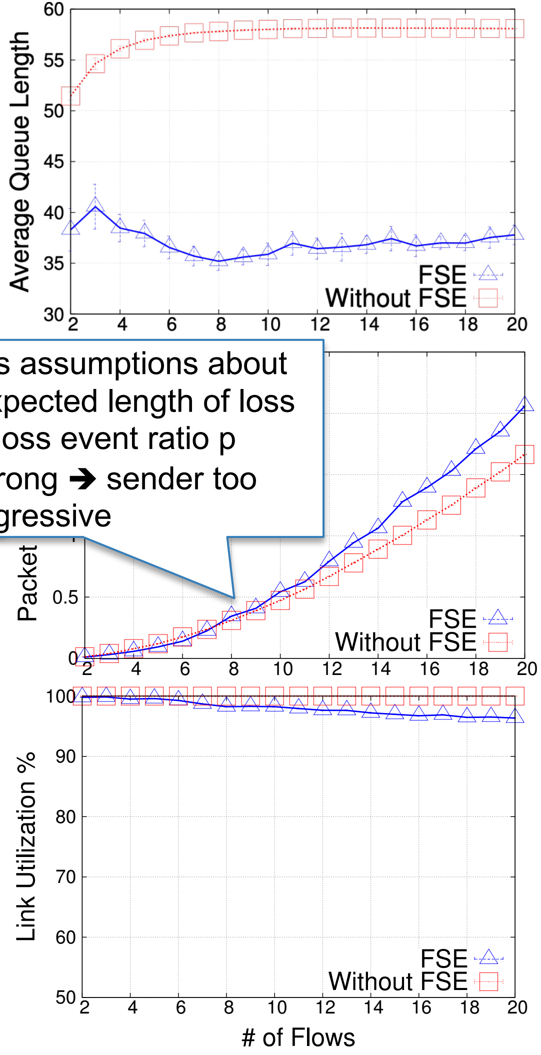
RAP **TFRC**



Average Queue

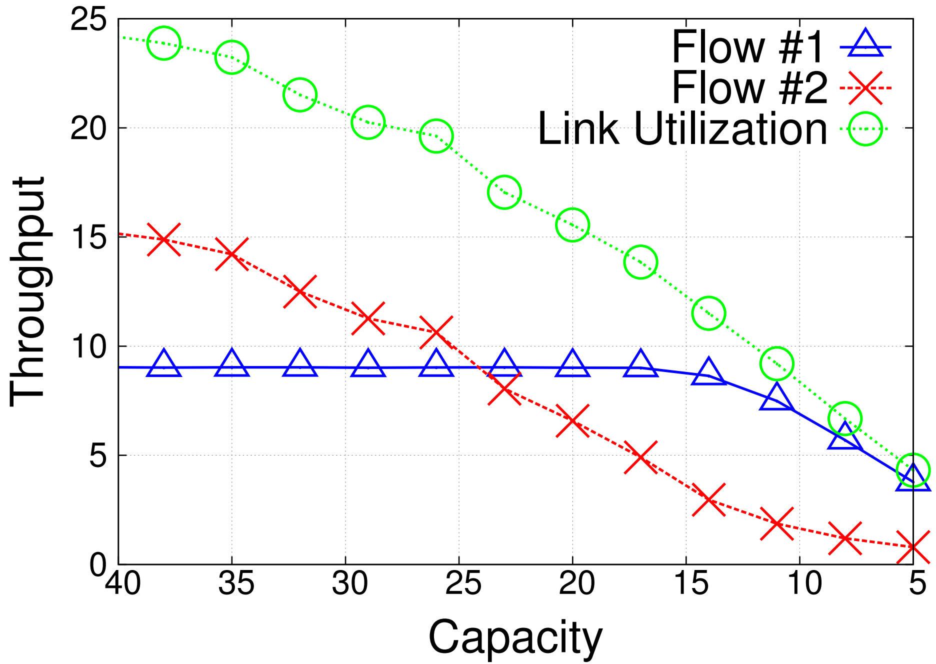
Packet Loss Ratio

Link Utilization



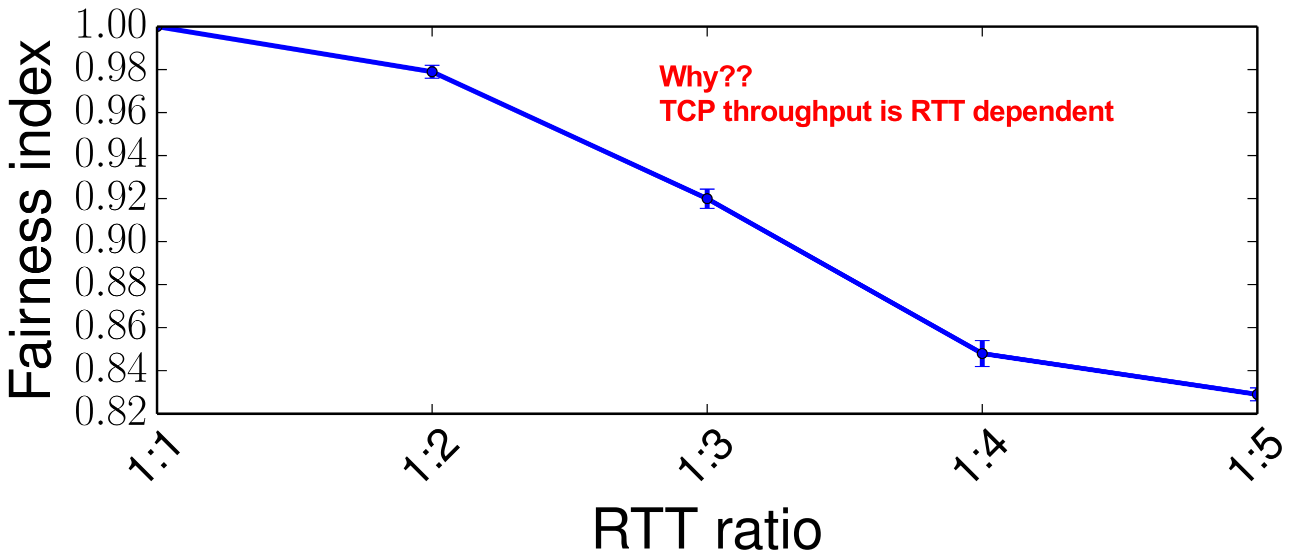
Receiver makes assumptions about sending rate (expected length of loss interval) ➔ loss event ratio p calculation wrong ➔ sender too aggressive

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

Why passive version doesn't always work: TCP example



Jains Fairness Index of 2 TCP Flows over a dumbbell topology

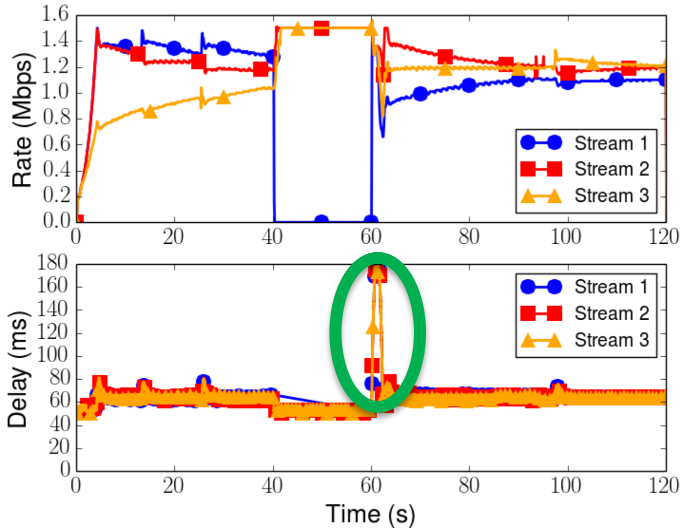
NADA and GCC

- ❑ RTT independent fairness
  - ➢ Rate update frequency
    - ☞ Fixed interval, RTT independent

-> Passive version works

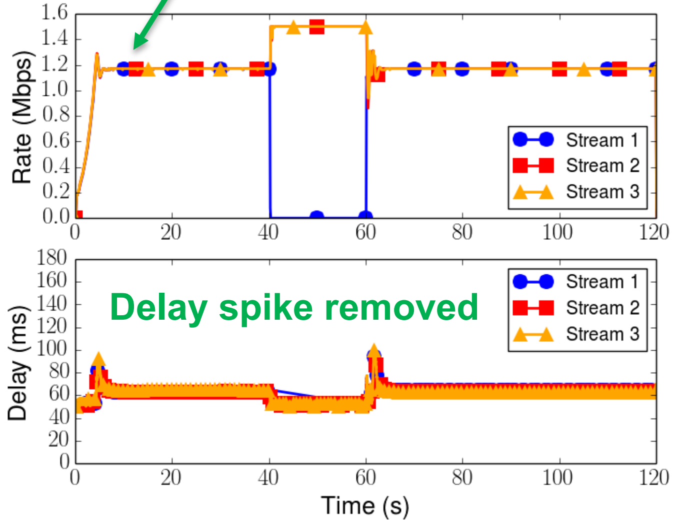
- ➢ Less signaling
- ➢ Simple request-response server
- ➢ Minimal changes
- ➢ Can work as a stand-alone tool

Media pause and resume (NADA)



Without FSE

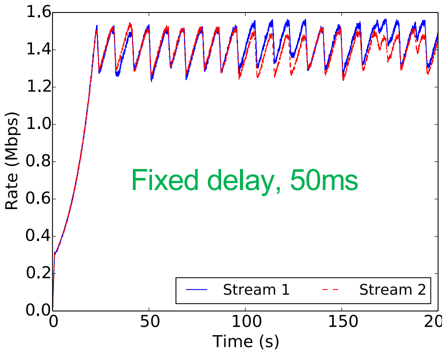
Faster convergence, Fairness

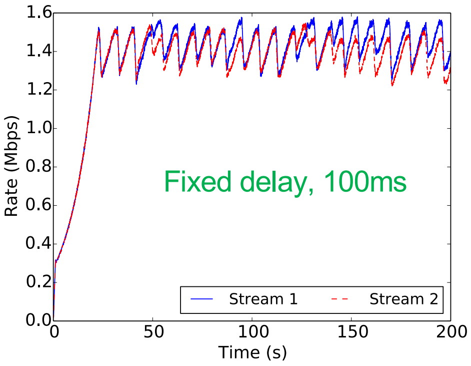


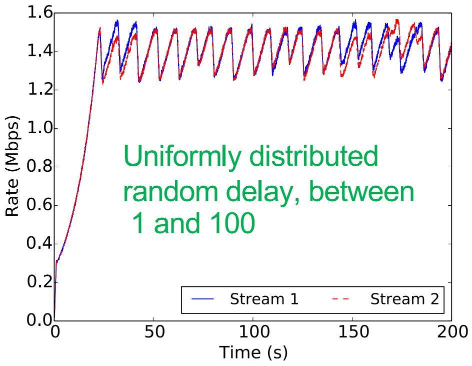
With FSE

Delayed feedback test (GCC)

- ❑ To show algorithm's robustness against OS's disturbance
  - Delay between stream 1 and the FSE is varied
  - Delay between stream 2 and the FSE is 0







Evaluations

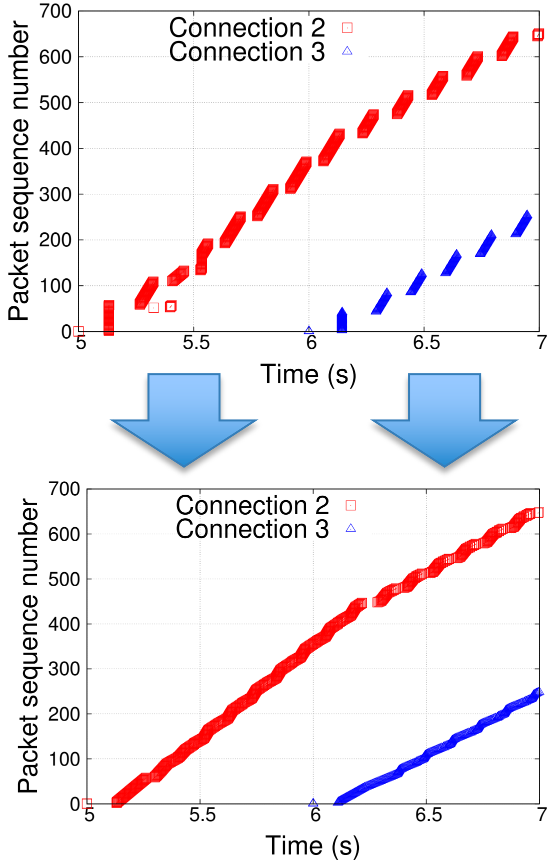
TCP congestion control coupling

- Novel ACK clock mechanism for initializing IW
- RFC 2140bis
- TCP-IN-UDP encapsulation
- TCP CCC
- Combining different ccc (LEDBAT and TCP)

ANRW'16
TCPM WG Draft
ICCRG Draft
GI'18
IMCEC'16

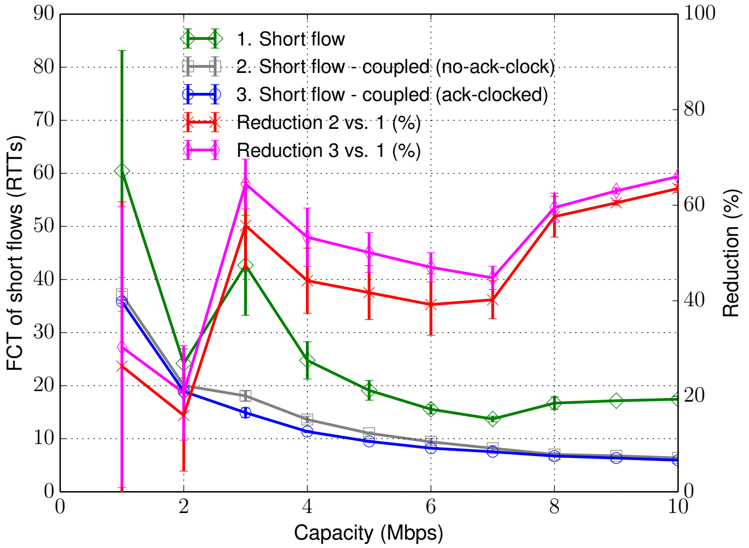
ACK-clocking to avoid bursts

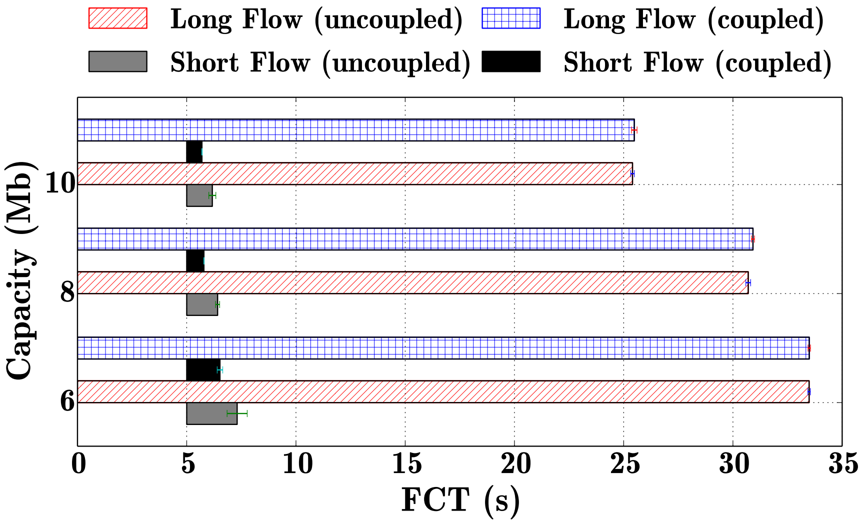
- A flow joining with a large share from the aggregate can create bursts in the network
  - *If* ***not paced***
- Our approach:
  - *Maintain the ack-clock of TCP*
  - *Using the ACKs of conn 1 to clock packet transmissions of connection 2 over the course of the first RTT when connection 2 joins*
  - *Similarly, we make use of the ACKs of connections 1 and 2 to clock packet transmissions of conn 3*
  - *Requires slightly more changes to the TCP code*



Drive an RFC2140 update to reflect the current state of the art, caveats on sharing TCB and pacing.

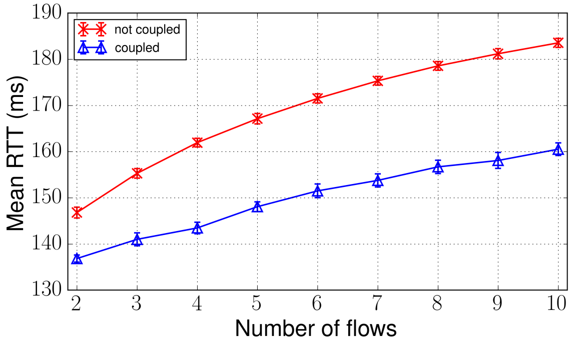
FCT of a short flow competing with a long flow



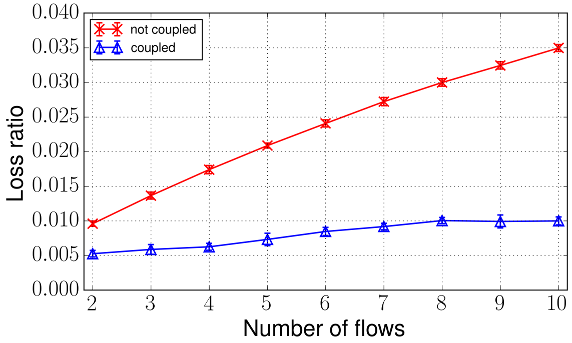


TCP-CCC (FreeBSD implementation)

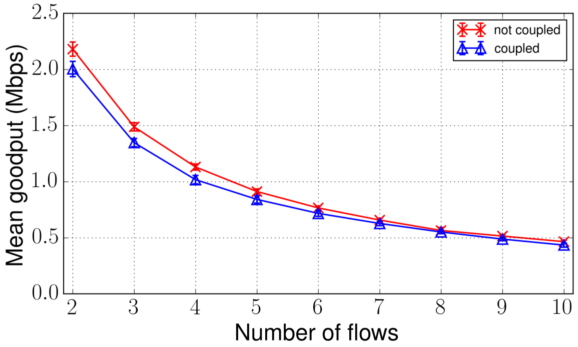
Avg. RTT



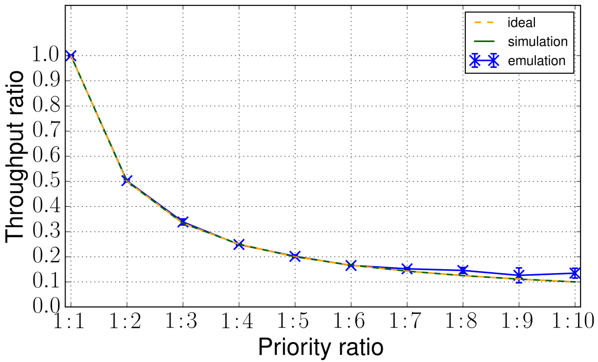
Loss ratio



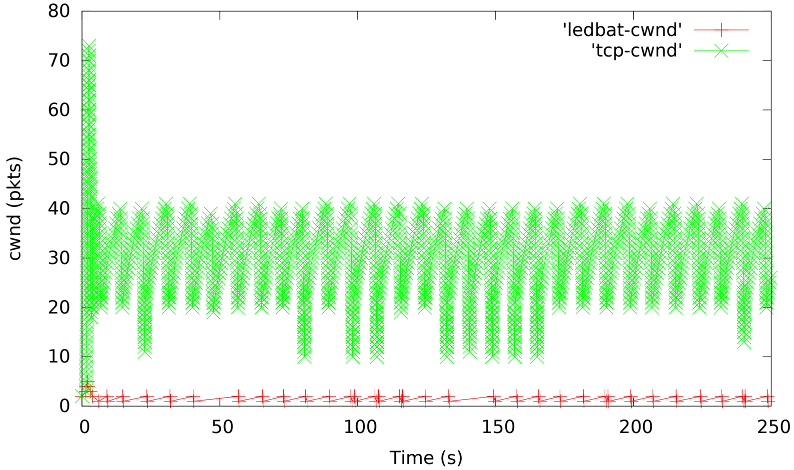
Avg. goodput



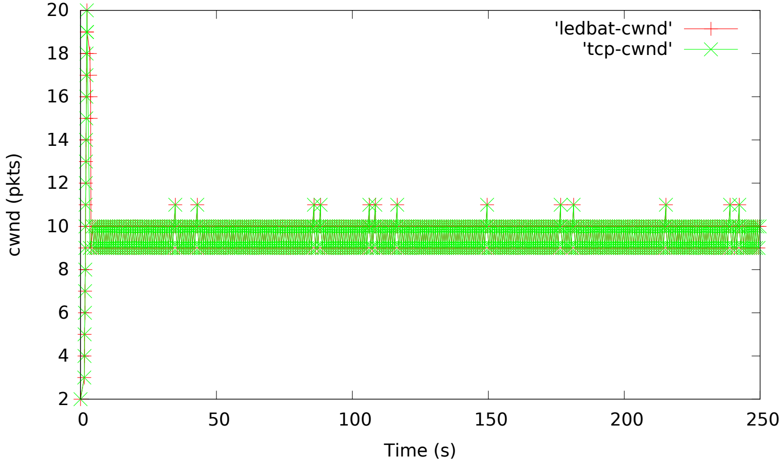
Prioritization



Combining Heterogeneous Congestion Controllers



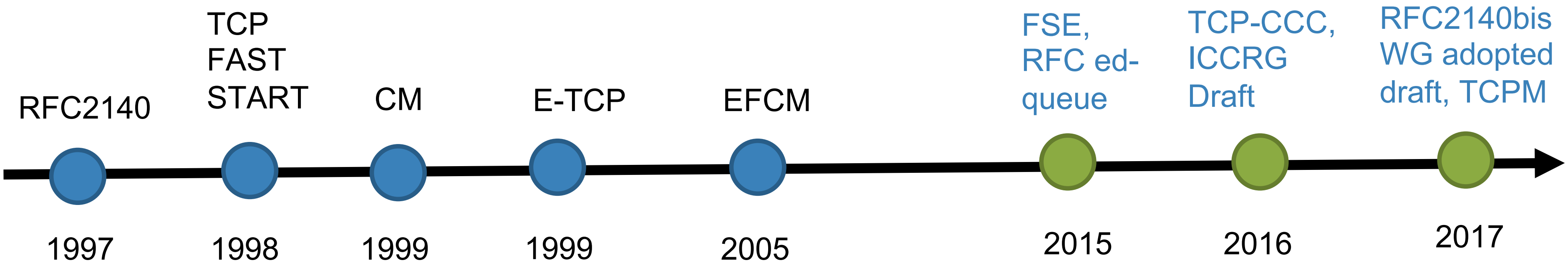
Without FSE



With FSE

This will allow us to combine WebRTC DATA Channel (SCTP) and Video (GCC) [Ongoing]

IETF Deployments



Source code available at: www.bitbucket.org/safiqul
http://safiquli.at.ifi.uio.no/coupled-cc

Implementations: 1) FreeBSD 2) Chromium Browser 3) ns-2

Changing algorithm aggression	Reducing aggression can improve performance (Paper-1), but there are exceptions: it can **violate the underlying CC algorithm's** assumption. This, in turn, can make the CC counteract on the imposed decision (paper-2 and draft-1).
RTT	Connections with homogeneous RTTs can use both active (paper 1) and passive coupling (paper-2, paper-4, paper-5). However, it is recommended to use an **active version** for connections with **heterogeneous RTTs** (paper 1).
Rate updates	Congestion control mechanisms that update their rates **not as a function of RTTs but e.g. at a fixed interval** can use **simple passive version** (paper 2).
Receiver- side Logic	If the CC decisions of a connection are influenced by **receiver-side CC logic**, this should be incorporated into the design of a coupled congestion control solution (paper 1).
Statefulness	It is recommended to **incorporate states** in a coupling solution when a congestion mechanism is stateful, e.g, TCP (paper 4,5 and draft 2). The design approaches for the stateless mechanisms are simpler (paper 1 and 2).
Ensured Common Bottleneck	Whenever it is **enforced that connections take a common path**, e.g., connections are multiplexed (e.g., WebRTC flows) or encapsulated (e.g., VPNs), a coupled congestion control mechanism can always be used (paper 1, 2,4,5 and draft 1,2).
Pacing	Giving a large share of the aggregate creates sudden bursts for window based congestion control, and therefore some form of pacing is required (paper 3). This can be achieved **with a timer or by gradually handing over the share of the aggregate**. Avoiding any increased burstiness due to CC coupling requires an algorithm to be active.
Combining Different CCs	Combinations of two different congestion control mechanisms can **avoid bad interaction**; for example, a loss-based controller can benefit from a delay-based controller which reacts on a congestion episode earlier (paper 4).

Thanks!









Q&A

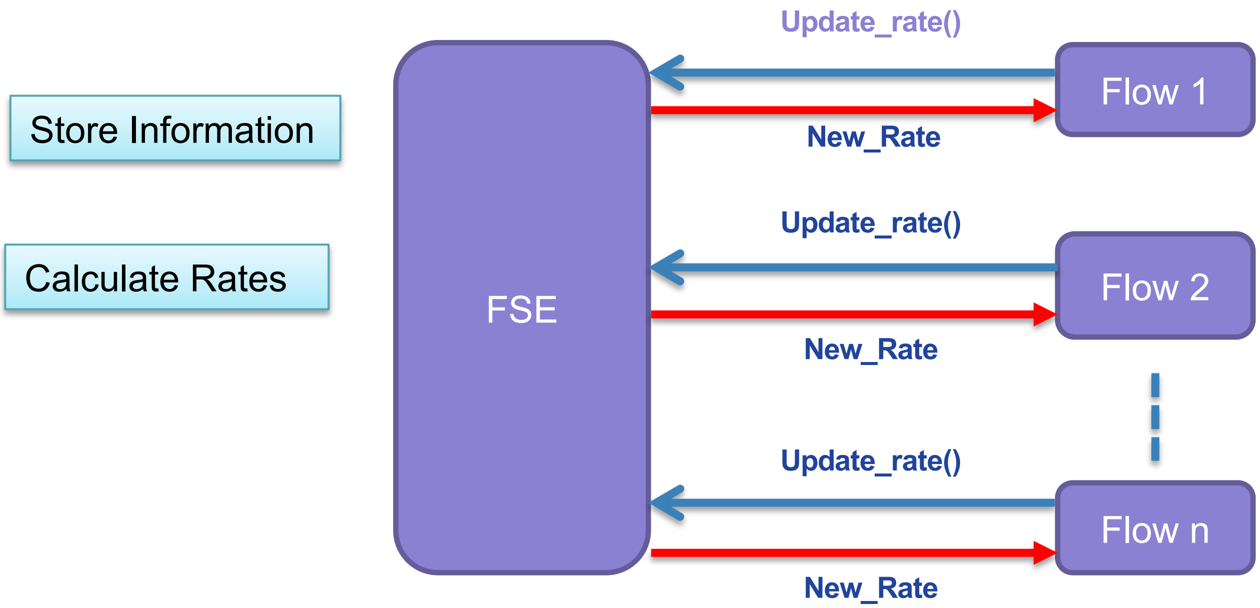
Paper	Publication
1	Safiqul Islam, Michael Welzl, Stein Gjessing, and Naeem Khademi, **Coupled congestion control for RTP media**, ACM Computer Communication Review, volume 44, Issue 4, October 2014
2	Safiqul Islam, Michael Welzl, David Hayes, Stein Gjessing, **Managing Real-Time Media Flows through a Flow State Exchange**, IEEE NOMS 2016, Istanbul, Turkey, 25-29 April 2016
3	Safiqul Islam, Michael Welzl, **Start Me Up: Determining and Sharing TCP's Initial Congestion Window**, ACM, IRTF, ISOC Applied Networking Research Workshop 2016 (ANRW 2016)
4	***OpenTCP: Combining Congestion Controls of Parallel TCP Connections,*** IEEE IMCEC 2016
5	Safiqul Islam, Michael Welzl, Kristian Hiorth, David Hayes, Greville Armitage, Stein Gjessing, **Single-Path TCP Congestion Control Coupling**, IEEE INFOCOM GI 2018
IETF ID-1	Safiqul Islam, Michael Welzl, Stein Gjessing, **Coupled Congestion Control for RTP Media, Internet-draft draft-ietf-rmcat-coupled-cc-06**, Mar 2017.
IETF ID-2	Michael Welzl, Safiqul Islam , Kristian Hiorth, Jianjie You: " TCP-CCC: single-path TCP congestion control coupling", Internet-draft draft-welzl-tcp-ccc-0, Oct 2016.
IETF ID-3	Joe Touch, Michael Welzl, Safiqul Islam, Jianjie You: TCP Control Block Interdependence, Internet-draft draft-touch-tcpm-2140bis-02, Jan 2017.

Backup Slides

Passive Coupling

#1

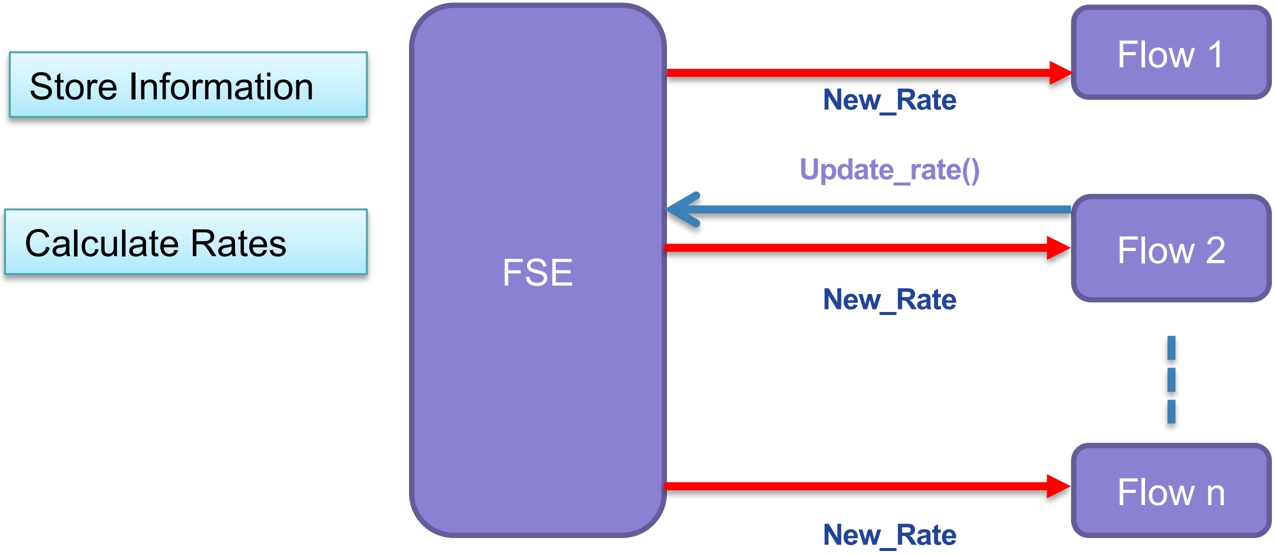
Simple and Flexible



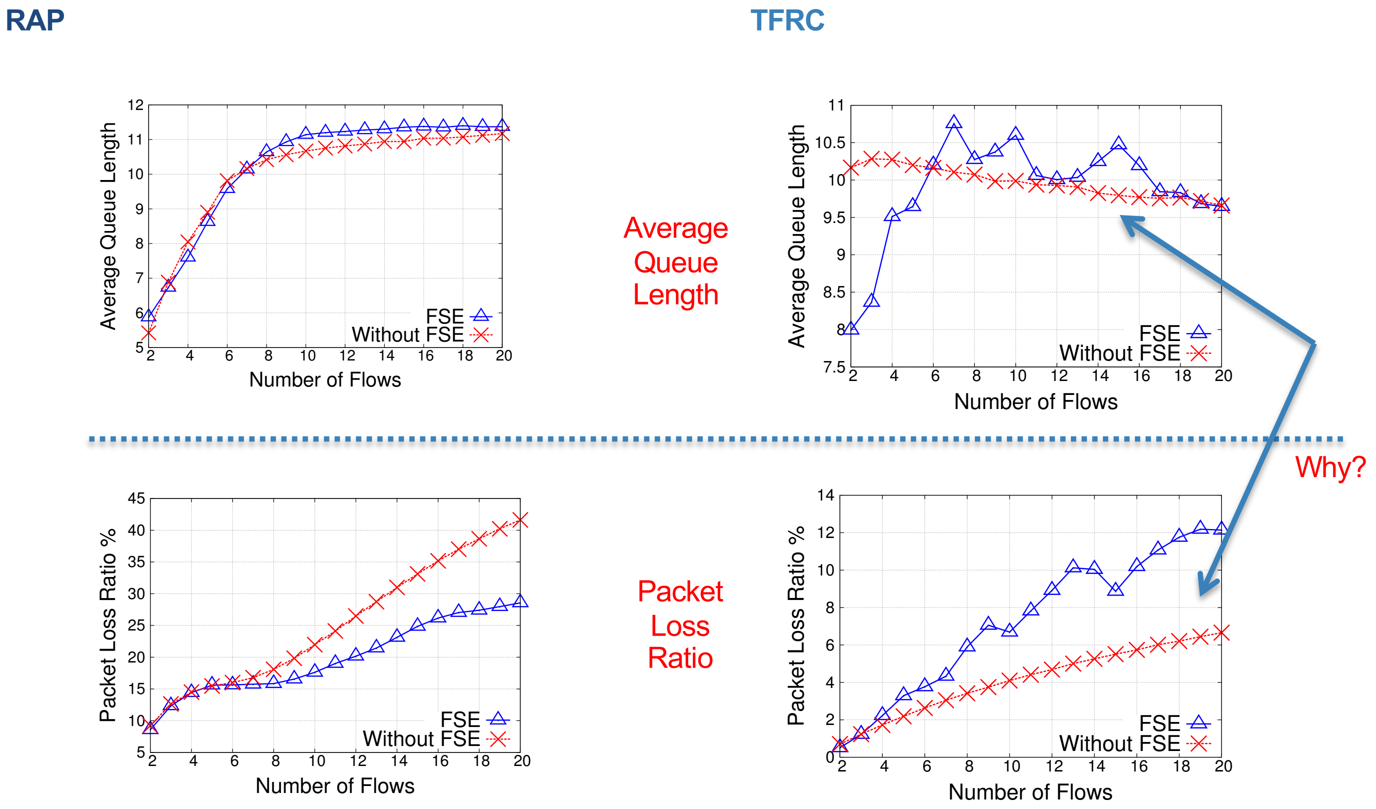
#1

Simple and Flexible

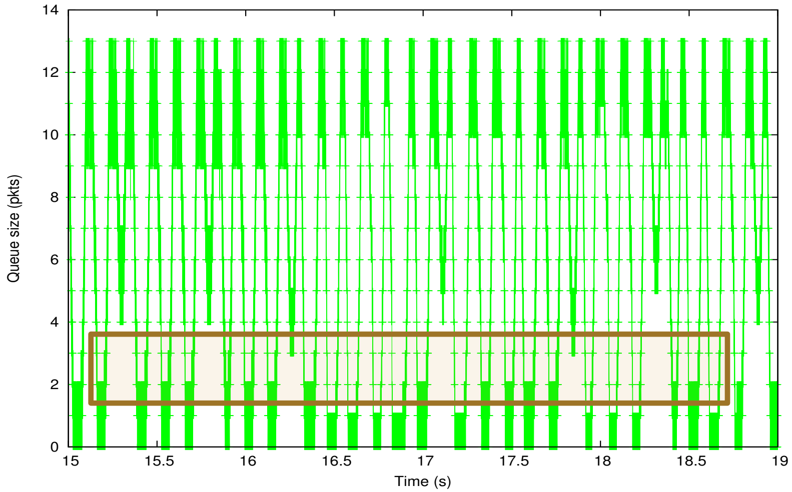
Active Coupling



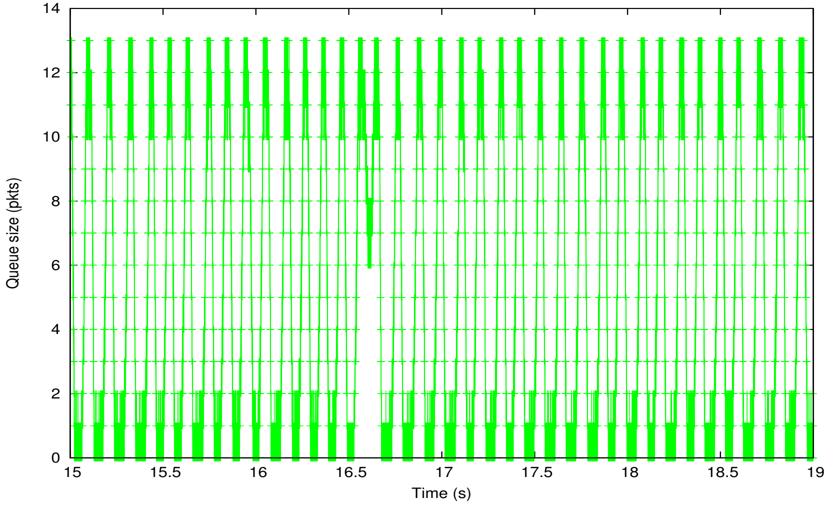
RAP and TFRC


RAP
TFRC
Average Queue Length
Number of Flows
FSE
Without FSE
Average Queue Length
Packet Loss Ratio
Why?
Packet Loss Ratio %
Number of Flows

What's going on? (simple algorithm)



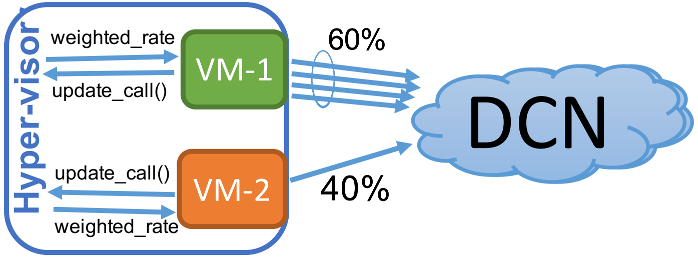
With FSE

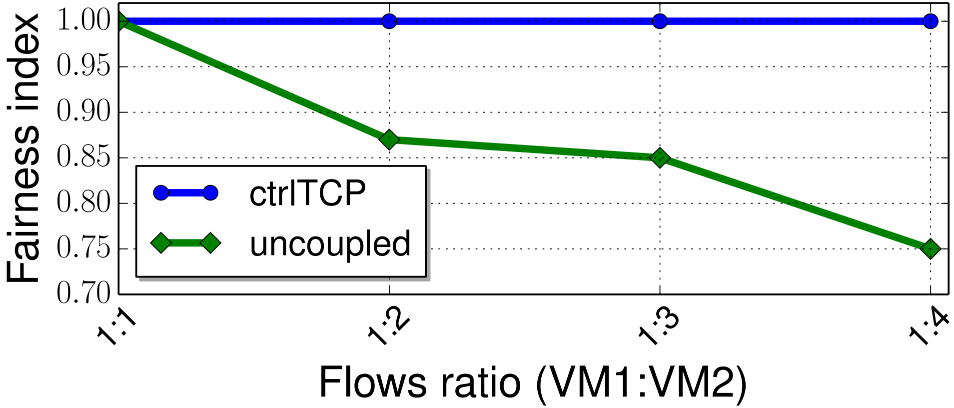


Without FSE

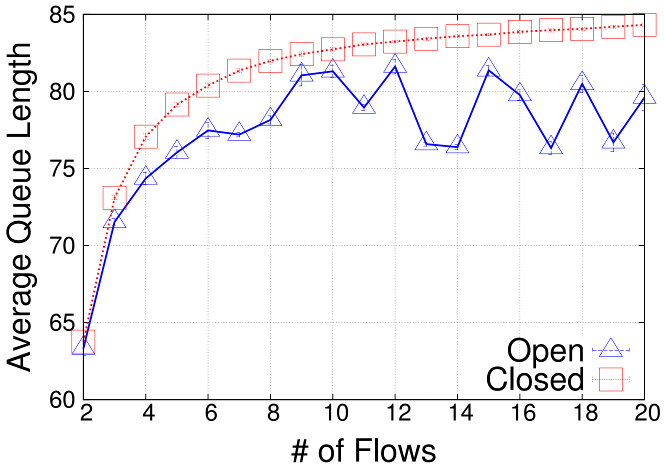
- ❑ Queue drains more often without FSE
- ➢ Should emulate the congestion response of one flow
  - ☞ FSE: 2 flows with rate X each; one flow halves its rate: 2X ➔ 1 ½X
  - ☞ When flows synchronize, both halve their rates on congestion, which halves the aggregate rate
  - ☞ We want that ! 2X ➔ 1X

CtrlTCP in DataCenter

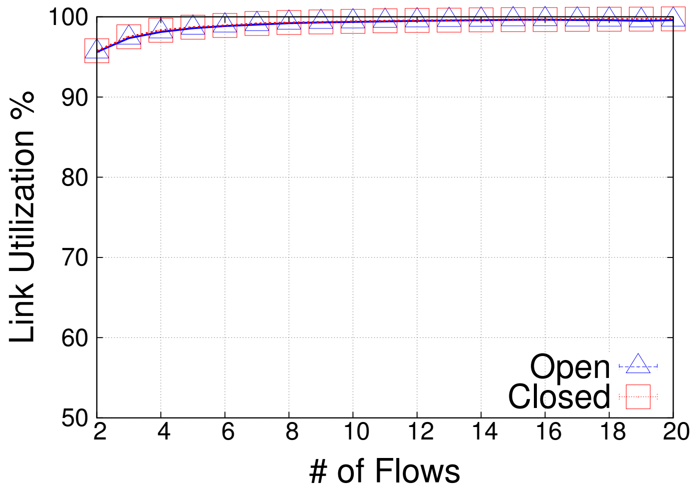




LEDBAT

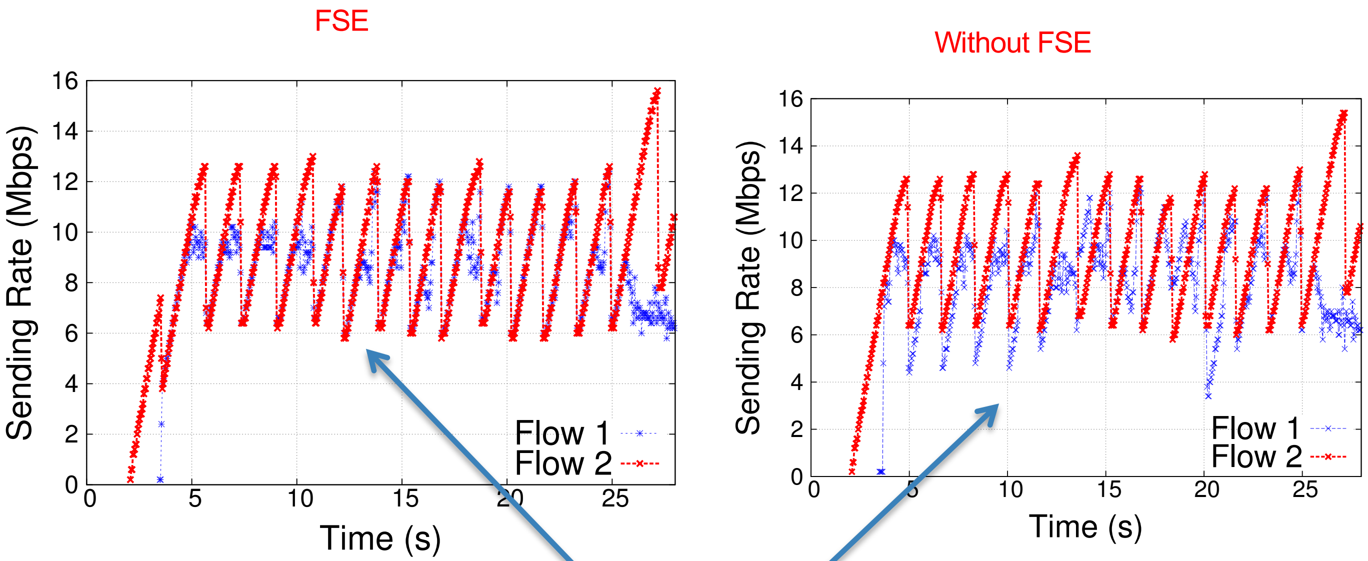


Average Queue Length
85
80
75
70
65
60
2
4
6
8
10
12
14
16
18
20
# of Flows
Open
Closed



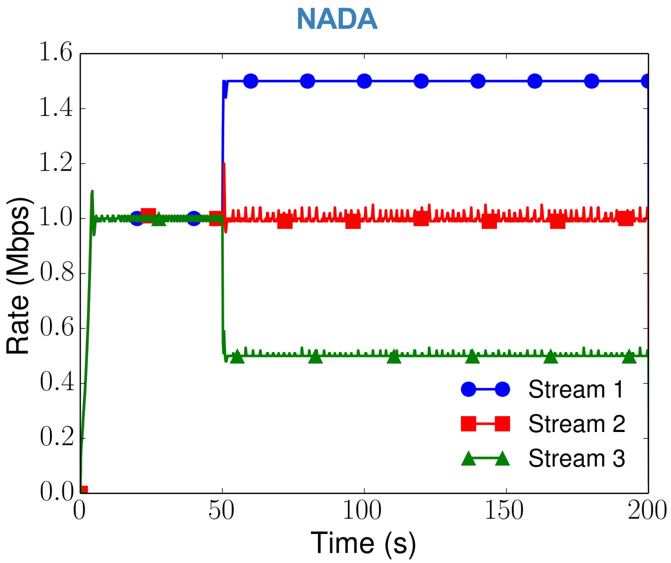
Link Utilization %
100
90
80
70
60
50
2
4
6
8
10
12
14
16
18
20
# of Flows
Open
Closed

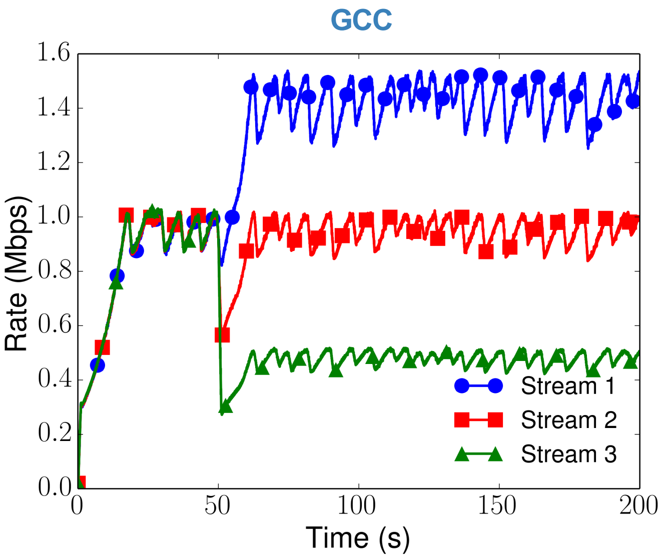
Evaluation – an application limited flow and one greedy flow (RAP)



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

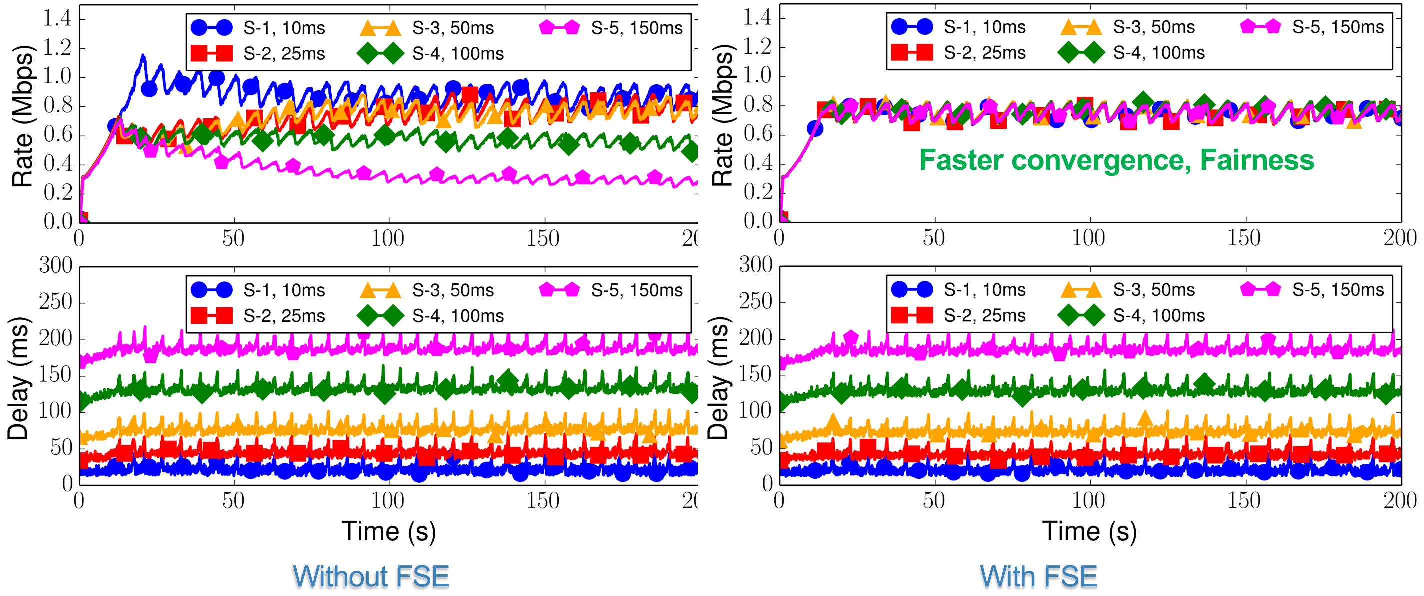
Prioritization test





One way base delay, 50ms, streams started with the same priorities.
Priorities are changed at 50 seconds.

Round-trip time fairness – GCC streams



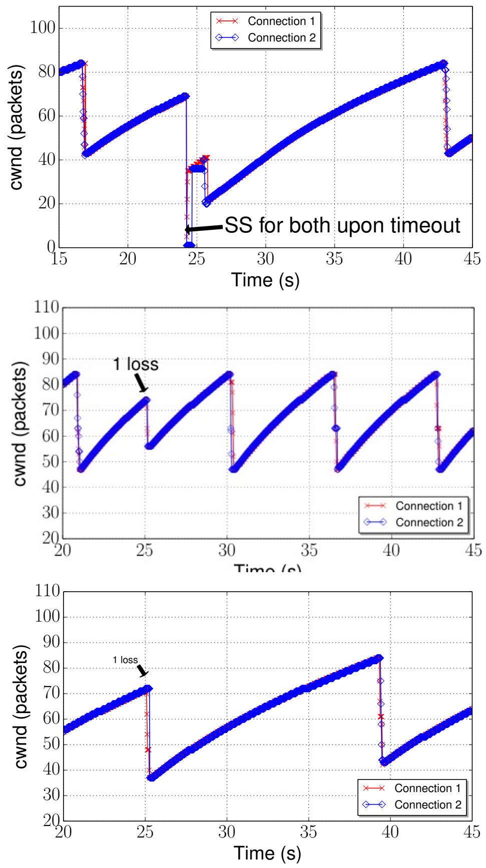
One-way delays of s1, s2, s3, s4 and s5 are 10ms, 25ms, 50ms, 100ms, and 150ms respectively, and bottleneck capacity 4Mbps

Design

- Basic idea similar to FSE in *draft-ietf-rmcat-coupled-cc*
  - *To emulate one flow's behavior (… but easy to tune)*
  - Keep a table of all current connections c with their priorities P(c); calculate each connection's share as P(c) / Σ(P) * Σ(cwnd); react when a connection updates its cwnd and use (cwnd(c) – previous cwnd(c)) to update Σ(cwnd)

TCP states

- ❑ Once in CA, Slow-Start(SS) shouldn't happen as long as ACKs arrive on any flow ➔ only SS when <u>all</u> flows are in SS
- ❑ Avoid multiple congestion reactions to one loss event:
  - ➢ TCP already has Fast Recovery (FR), use that instead



E-TCP

EFCM

Basic TCP changes

The required changes to TCP:

- This function call, to be executed at the beginning of a TCP connection ‘c’ :

  register(c, P, cwnd, sshtresh);

  returns: cwnd, ssthresh, state

- This function call, to be executed whenever TCP connection ‘c’ newly calculates cwnd:

  update(c, cwnd, sshthresh, state);

  returns: cwnd, ssthresh, state

- This function call, to be executed whenever a TCP connection ‘c’ ends:

  leave(c)