RMCAT architectural overview

Michael Welzl
michawe@ifi.uio.no

RMCAT, 85th IETF Meeting
8. 11. 2012

Disclaimer

- I'll be talking about "sender" and "receiver" here, just to differentiate roles
  - Yes we're dealing with bidirectional traffic, but so is TCP, and talking about "sender" and "receiver" roles never was a problem there

A framework

- Delay-based congestion control has many, many issues
  - Unlikely that we solve them all straight away
- Charter: “Determine if extensions to RTP/RTCP are needed for carrying congestion control feedback, using DCCP as a model. If so, provide the requirements for such extensions to the AVTCORE working group for standardization there.”
  - This sounds like “define fields”, but DCCP had to do much more to become a framework
  - It may also be overloaded...
- The framework involves some general design decisions that would affect all cc. mechanisms we standardize
  - Better get them right from day 1

TCP, for example

- Feedback
  - Unreliable ACKs; can lead to misinterpretation of backward loss as forward congestion
  - not a big deal because information in ACKs redundant, and lots of ACKs are sent
  - to detect backward congestion (and do ACK cc.), sender must know receiver's ACK ratio
- Reaction to ECN
  - MUST be similar to reaction to loss, for compatibility with non-ECN-capable TCP flows
- Pluggable congestion control
  - even without standardization, sender-side change, no need to even inform the receiver
  - possible because of rather "dumb" (better: "generic") receiver

Lessons learned from TCP

- Feedback
  - To avoid misinterpreting feedback loss as forward congestion and/or do backwards congestion control: consider making ACKs reliable (see DCCP)
- Reaction to ECN
  - We're designing stuff from scratch here, could make all RMCAT flows ECN-capable => no need to protect non-ECN-capable RMCAT flows (?)
  - Flows that don't get ECN marks from the same bottleneck: not likely (?)
- Pluggable congestion control
  - Probably desirable
  - Requires one side to be generic or [Randell]: both sides generic, might be possible to exchange either one of them

Devil's advocate: Consider RRTCC (draft-alvestrand-rtcweb-congestion-03), for example

- Receiver:
  - Look at changes in inter-packet delay, apply some maths (Kalman filter)
  - If the sender should stop increasing (or a feedback timer expires), tell it "your new rate is X"

- Sender:
  - calculate TFRC equation; rate is max (result of TFRC equation, X)
  - In the absence of a feedback, increase rate
    (but: no feedback for a long time => timeout)

- Can we agree that:
  - this receiver behavior is ok for all possible future senders?
  - this sender behavior is ok for all possible future receivers?

- Thinking of TCP again:
  the simpler one side, the more flexible the other becomes

Sender- vs. Receiver-based

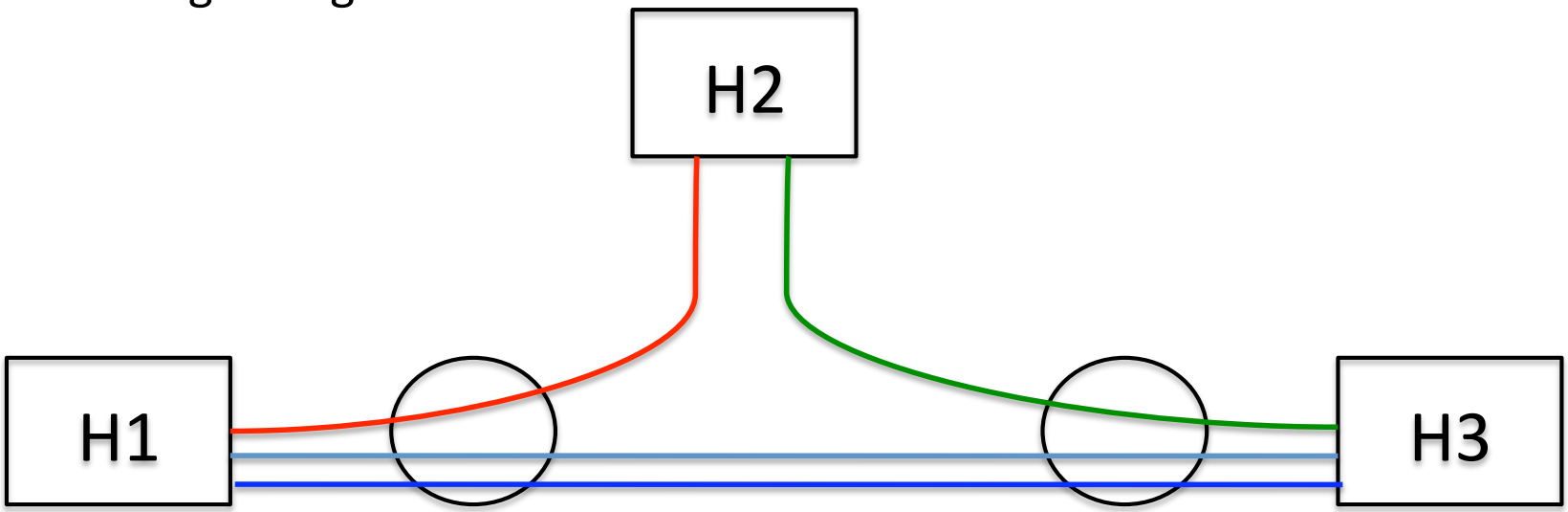
- Perhaps we should decide now which side to make simple?

- Many sides to sender- vs. receiver-based CC... Key question: *always* minimize feedback or not?
  - − might make the control unnecessarily fragile
  - + less traffic is less traffic... also: simpler than feedback-CC, and e.g. smaller chance of collision on wireless

- Many more pro's and con's... e.g., "interactions with applications": importance of packets in send buffer could play a role for congestion control decision
  - – affects signalling in case of receivers-side cc.

Coupled congestion control

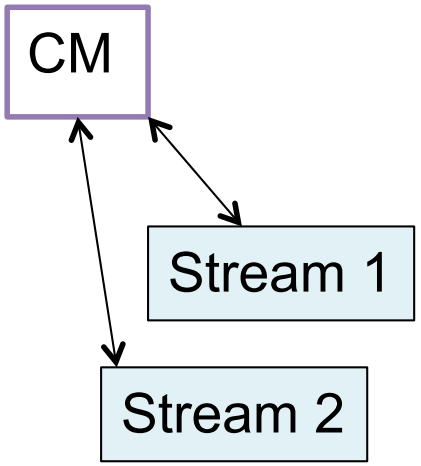
- Only makes sense for flows that share a bottleneck
  - Obvious *for some flows* in case of WebRTC (same 5-tuple = same bottleneck)
  - Less so in the general case, but there are working methods
  - Coordinate streams between multiple hosts: need to detect shared bottlenecks on both sides
    - Signalling needed



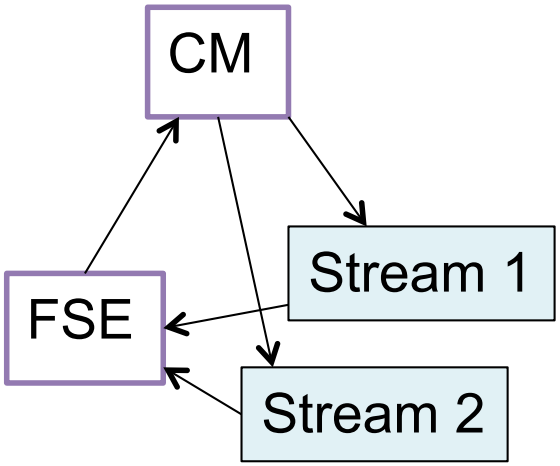
“Flow State Exchange” (FSE)

- The result of searching for minimum-necessary-standardization: only define what goes in / out, how data are maintained
  - Could reside in a single app (e.g. browser) and/or in the OS

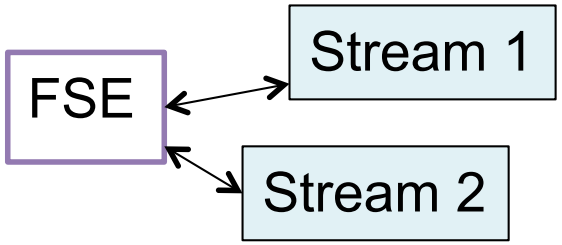
Traditional CM



FSE-based CM



Another possible implementation of flow coordination



Thank you!

Questions?