RMCAT - Shared Bottleneck Detection and Coupled Congestion Control vs. QoS for WebRTC

Michael Welzl
Contributors

• Main people behind this work:
  – Shared bottleneck detection:
    Algorithm development & evaluation: **David Hayes**
    Stuff available from: [http://heim.ifi.uio.no/davihay/](http://heim.ifi.uio.no/davihay/)
    – NorNet testing: **Simone Ferlin** (Simula Research Labs)
  – Coupled congestion control:
    **Safiqul Islam**
    Stuff available from: [http://heim.ifi.uio.no/safiquli/coupled-cc/index.html](http://heim.ifi.uio.no/safiquli/coupled-cc/index.html)
    – Other contributors: **Stein Gjessing, Naeem Khademi**
WebRTC in a nutshell

• Multimedia services in browsers, available everywhere, without plugins
  – P2P (browser-to-browser); major focus on traversing middleboxes (NATs etc) ⇒ all traffic will be one UDP 5-tuple
  – Includes SCTP data channel (e.g. control traffic in a game)
  – Also a lot of IETF effort to agree on audio/video codecs

• Control by web developer, via Javascript API, standardized in W3C

• Protocols standardized in IETF WGs: RTCWEB (main) + RMCAT (congestion control) + DART (QoS)
  – Note, RMCAT charter is about media, not only WebRTC
WebRTC possibilities: games, toys... and talking to tech support in the browser
Prioritization

• WebRTC needs priorities
  – e.g., draft-ietf-rtcweb-use-cases-and-requirements includes use case “Simple Video Communication Service, QoS”
  – Priorities to be exposed via JS API

• Just use QoS?
  – Not available everywhere; some people just want to set DSCP and “hope for the best”
  – Needs special rules: different DSCP values for one UDP-5-tuple?! draft-ietf-dart-dscp-rtp-10
  – When common shared bottleneck is known, priorities between flows from a single sender realized by coupled congestion control (QoS can still protect you from other users)
Coupled congestion control: background

- Having multiple congestion controlled flows from the same sender compete on the same bottleneck is detrimental
  - Combine congestion controllers in the sender $\rightarrow$ better control fairness (with priorities) and get less delay and loss

- Two elements: 1) shared bottleneck detection (sbd), 2) coupled congestion control
  - In WebRTC, 1) can sometimes be very easy: same 6-tuple. But measurement-based sbd enables broader application of 2) (same sender, different receivers)
  - apparently, “same 6-tuple” turned this into “QoS vs. coupled-CC” in the IETF
Shared bottleneck detection: scenarios

- Only case 1 can be supported by 6-tuple
- Case 2 can be supported by measurement-based SBD
- Case 3 requires coordinating senders H1 and H2
  - Can be supported by measurement-based SBD, but needs more (H1 ↔ H2 coordination); not currently considered
SBD for RMCAT

[ draft-hayes-rmcat-sbd-01.txt ]

• Requirements:
  – Little signalling (also true for RMCAT CC algos)
  – Passive: require no changes to traffic
  – Reliable for long-term data/multimedia streams (e.g. video)
    • False positive worse than false negative
  – Realistic to implement as a real-time function in a browser
    (no expensive offline computation)
SBD overview

• Goal: determine correlation from common queue

• Receiver calculates summary statistics from OWD
  – To deal with noise, lag and limit feedback

• **Variance**: Packet Delay Variation [RFC 5481]
• **Skewness**: skew_est
• **Oscillation**: freq_est (avg. # of significant mean crossings)

• Sender: group flows experiencing congestion (skew_est); divide based on freq_est; PDV; skew_est
Why skewness?

- Change from positive to negative near 100% load makes it a good congestion indicator.
- Not good to use alone: ambiguity and susceptible to extreme sample values.
Practical estimation of skewness

- Count proportion of samples above and below the mean
- Removes ambiguity
Simulation setup

- Background traffic based on real traffic traces
  - >90%
- Flows 1&2 send at twice the rate of 3&4
- Various combinations of bottlenecks activated
Simulation

Queue sizes subsampled (1:200) and OWDs (1:10)

Path delays ms
link 10ms std 2.0
86.2, 92.1, 93.6, 87.4
Real network tests with NorNet

Background Traffic
- A Exponential
- B & C LRD (Hurst=0.8)
- Bottlenecks mostly on 3G link
- majority

Application Traffic
- D, E, & F exp.
- G CBR
- minority (<10 %)

https://www.nntb.no/
Coupled congestion control

• When possible, best done by scheduling packet transmission from different sources with a single congestion controller
  – Congestion Manager (CM)

• Disadvantages of this approach:
  – Hard to combining multiple applications (RMCAT is about RTP-based applications in general, not only WebRTC)
  – Difficult to switch on/off

• Hence, goal of [draft-welzl-rmcat-coupled-cc-04]: achieve benefits with minimal changes to existing congestion controllers

Best paper award ➔ to be published in ACM SIGCOMM CCR too.
“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
First version: only passive

- Goal: **Minimal change to existing CC**
  - each time it updates its sending rate (New_CR), the flow calls update (New_CR, New_DR), and gets the new rate
  - Complicates the FSE algorithm and resulting dynamics (e.g. need dampening to avoid overshoot from slowly-reacting flows)
Now: FSE - Active

- Actively initiates communication will all the flows
  - Perhaps harder to use, but simpler algorithm and “nicer” resulting dynamics
Now: FSE - Active

- Actively initiates communication will all the flows
  - Perhaps harder to use, but simpler algorithm and “nicer” resulting dynamics

```
Store Information
Calculate Rates
```

```
FSE
Flow 1
New_Rate
Update_rate()
New_Rate
Flow 2
New_Rate
Flow n
```
Active algorithm: 1\textsuperscript{st} version

- Every time the congestion controller of a flow determines a new sending rate CC\_R, the flow calls UPDATE
  - Updates the sum of all rates, calculates the sending rates for all the flows and distributes them to all registered flows

\begin{verbatim}
for all flows i in FG do
    FSE\_R(i) = (P(i)\times S\_CR)/S\_P
    send FSE\_R(i) to the flow I
end for
\end{verbatim}

- Designed to be as simple as possible
Evaluation

• Done: simulations, rate-based
  – AIMD: RAP
  – Multimedia-oriented: TFRC

• Ongoing: simulations, window-based:
  – Delay-based: LEDBAT
  – TCP (SCTP-like, for data channel)

• Planned: currently proposed RMCAT CC’s
  – Simulations
  – Real-life tests (put code in browser)
Simulation setup

- 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%)
Dynamic behavior: Rate Adaptation Protocol RAP ( = rate-based AIMD)

With FSE

Without FSE
Dynamic behavior: TFRC

With FSE

Without FSE
FSE goals

- Charter:
  “Develop a mechanism for identifying shared bottlenecks between groups of flows, and means to flexibly allocate their rates within the aggregate hitting the shared bottleneck.” (requirement F34 in draft-ietf-rtcweb-use-cases-and-requirements-12)
  - This works perfectly

- But: because this avoids competition between flows, we expected reduced queuing delay and loss as a side effect
Average queue length (RAP)

![Graph showing the relationship between average queue length and number of flows with and without FSE]
Packet loss ratio (RAP)
What’s going on?

- Queue drains more often without FSE
  - E.g.: 2 flows with rate $X$ each; flow 1 halves its rate: $2X \rightarrow 1 \frac{1}{2}X$
  - When flows synchronize, both halve their rate on congestion, which really halves the aggregate rate: $2X \rightarrow 1X$
Fix: Conservative Active FSE algorithm

• No congestion (increase): do as before
• Congestion (decrease): proportionally reduce total rate (like one flow)
  – e.g. flow 1 goes from 1 to ½ => total goes from X to X/2

• Need to prevent that flows ignore congestion or overreact
  – timer prevents rate changes immediately after the common rate reduction that follows a congestion event
  – Timer is set to 2 RTTs of the flow that experienced congestion
  – Reasoning: assume that a congestion event can persist for up to one RTT of that flow, with another RTT added to compensate for fluctuations in the measured RTT value
Receiver makes assumptions about sending rate (expected length of loss interval) \( \Rightarrow \) loss event ratio \( p \) calculation wrong \( \Rightarrow \) sender too aggressive
Multiple LEDBAT flows

Average Queue Length of LEDBAT Flows – with and without FSE

Link Utilization of LEDBAT Flows – with and without FSE
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
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 application 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.
2 FSE controlled flows competing with synthetic traffic (TFRC)

- 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, adapted to provide an approximate load of 50%

Throughput ratios very close to theoretical values ➔ FSE operation largely unaffected
Thank you!

Questions?