Coupled Congestion Control for RTP Media

Safiqul Islam, Michael Welzl, Stein Gjessing and Naeem Khademi
Department of Informatics, University of Oslo, Norway
{safiquli | michawe | steing | naeemk}@ifi.uio.no

ABSTRACT

Congestion occurs at a bottleneck along an Internet path; multiple flows between the same sender and receiver pairs can benefit from using only a single congestion control instance when they share the same bottleneck. These benefits include the ability to control the rate allocation between flows and reduced overall delay (multiple congestion control instances cause more queuing delay than one since each has no knowledge of the congestion episodes experienced by the others). We present a mechanism for coupling congestion control for real-time media and show its benefits by coupling multiple congestion controlled flows that share the same bottleneck.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications—*Computer conferencing, teleconferencing, and videoconferencing*

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Congestion Control; FSE; RMCAT; WebRTC

1. INTRODUCTION

Multiple congestion controlled flows (e.g., TCP) between the same two hosts usually have separate congestion control instances, even when the path used by them is the same. There may be several reasons for this separation. For example, one cannot always be sure if the path is indeed the same – routing mechanisms like Equal-Cost Multi-Path (ECMP) may assign different flows to different paths to achieve load balancing, even when they have the same destination IP address.

Routers or other middle-boxes usually identify flows using a five-tuple of source and destination IP addresses, transport protocol, and the transport protocol's source and destination port numbers. When – as it will be possible with the new WebRTC standard for interactive communication between web browsers – multiple flows are multiplexed over a single UDP port pair, they are normally regarded as a single flow inside the network and therefore treated in the same way. In such a setup, congestion management can be readily applied.

The new "RTP Media Congestion Avoidance Techniques" (RMCAT) IETF Working Group develops standards for RTP-based interactive real-time media. WebRTC being the major use case for these standards, RMCAT will also standardize methods for coupled congestion control, with the goal of having the best possible control over the send rate allocation. Here, we describe the first proposal for RMCAT's coupled congestion control and show its feasibility and some of its benefits.

After a review of related work in the next section, we will introduce our method for coupling congestion control in RMCAT in Section 3. In Section 4, we show some performance evaluation results using ns-2 simulations, and Section 5 concludes the paper.

2. RELATED WORK

The Congestion Manager (CM) [3] is the best known, and perhaps the oldest related work. It provides a common congestion management framework for all the flows from a sender going to the same receiver. Flows pass information to the CM which uses a scheduler to distribute the available bandwidth. Since the CM replaces each flow's congestion controller with an overarching one, it is hard to implement, which may be the reason why it has never been widely deployed.

In any standard TCP implementation, each connection maintains state (e.g. the current round-trip time (RTT) and congestion window (*cwnd*)) in a data structure called Transport Control Block (TCB). RFC 2140 [12] describes that TCB data can be shared among multiple connections in two ways: 1) Temporal Sharing, and 2) Ensemble Sharing. Temporal Sharing can be used to cache state of a closed connection, and this previous connection state can be used to later instantiate a similar connection and avoid inefficiencies. Ensemble Sharing occurs when an active host opens another concurrent connection. Among other variables, RFC 2140 discusses how *cwnd* can be shared in order to couple the congestion control of multiple flows.

Ensemble TCP (E-TCP) [4] utilizes the concept of TCB information reusing and sharing among existing connections. It has been designed to show the aggregate network trans-

CSWS'14, August 18, 2014, Chicago, IL, USA.
Copyright 2014 ACM 978-1-4503-2991-0/14/08 ...$15.00.
http://dx.doi.org/10.1145/2630088.2630089.

mission behavior of an ensemble (parallel TCP connections) as a single TCP/Reno connection. The authors of [4] compared it with persistent HTTP 1.1, showing benefits. E-TCP does not discuss what RFC 2140 calls Temporal Sharing, i.e. reusing cached information when the network is idle because network properties might change during an idle period.

Based on E-TCP, Savoric et al. [11] proposed an Ensemble Flow Control Mechanism (EFCM) where a controller actively probes for information from the flows, and calculates the new rate for a flow by aggregating congestion properties (e.g. RTT, *cwnd*). They showed that EFCM increases the throughput and fairness for the flows sharing the same bottleneck.

Both E-TCP and EFCM are similar in style to the mechanism presented in this paper. However, there are some important differences: these mechanisms focus exclusively on TCP congestion control, which is window based, whereas our mechanism targets rate-based RTP applications. Neither [4] nor [11] present an evaluation of the mechanism's impact on queuing delay or packet loss; reducing both is an important goal for us (RMCAT targets low-latency interactive applications). Since we tried to minimize changes needed to existing congestion controls, we only share rates between flows, whereas E-TCP and EFCM share not only *cwnd* but also other TCP-specific information such as $SRTT$ and *ssthresh*.

Rather than trying to directly combine the congestion control of multiple flows, a similar behavior can also be attained by multiplexing application-level data streams onto a single connection. This can be done using e.g. SCTP, where it can lead to a significant performance benefit [9]. In [15], a performance gain was attained by transparently mapping TCP connections onto a single SCTP association. Connection reuse – with the goal of allowing TCP's congestion window to grow larger and reduce transport-layer overhead – can also be implemented at the application layer, e.g. via persistent HTTP 1.1. However, HTTP 1.1 only allows delivery of application-level streams in the sequence in which they were requested, which can cause Head-Of-Line (HOL) blocking, e.g. when the first request involves a slow database access. This has recently been addressed by SPDY, which *multiplexes* data streams onto a single TCP connection [1].

3. THE FLOW STATE EXCHANGE

RMCAT's congestion control should be applicable but not limited to WebRTC. This means that we may need to jointly control flows that reside within a single application (a web browser, in case of WebRTC) or in multiple applications. In the latter case, WebRTC's benefit of knowing that packets from multiple flows will be routed in the same way is lost. There are, however, measurement based methods to determine whether multiple flows share a bottleneck in the network; being able to make use of measurements when necessary, and supporting various intra- as well as inter-application scenarios calls for a congestion management architecture that is much simpler than, e.g., the well-known CM.

We have opted for an approach [14] that minimizes the amount of necessary changes to existing applications. It involves a central storage element called "Flow State Exchange" (FSE). The elements of our architecture for coupled congestion control are: the Flow State Exchange (FSE), Shared Bottleneck Detection (SBD) and Flows. The FSE is a storage element that can be implemented in two ways: *active* and *passive*. In the active version, it initiates communication with flows and SBD. However, in the passive version, it does not actively initiate communication with flows and SBD, and its only task is internal state maintenance (e.g., an implementation could use soft state to remove a flow's data after long periods of inactivity).

Every time a flow's congestion control mechanism would normally update its sending rate, the flow instead updates information in the FSE and performs a query on the FSE, leading to a sending rate that can be different from what the congestion controller originally determined. In the active version, the FSE additionally calculates the rates for all the other flows in the same Flow Group (FG) and actively informs their congestion controllers with a callback function. A Flow Group consists of flows which should be controlled together, i.e. they have a common network bottleneck. A FG is determined by an SBD module based on measurements or knowledge about multiplexing. An SBD module can be a part of one of the applications using the FSE, or it can be a standalone entity. We plan to develop a measurement-based SBD as future work; in this paper, we assume that FGs are known by multiplexing flows over the same UDP port pair in WebRTC.

The FSE contains a list of all flows that have registered with it. For each flow, it stores a unique flow number to identify the flow, the Flow Group identifier that it belongs to, a priority P, and the calculated rate FSE_R. Each FG contains one static variable S_CR which is the sum of the calculated rates of all flows in the same FG.

Flows register themselves with SBD and FSE when they start, deregister from the FSE when they stop, and carry out an UPDATE function call every time their congestion controller calculates a new sending rate. Via UPDATE, they provide the newly calculated rate. The FSE then calculates rates for all the flows and sends them back. When a flow f starts, FSE_R is initialized with the congestion controller's initial rate. SBD will assign the correct FGI. When a flow is assigned an FGI, it adds its FSE_R to S_CR. When a flow stops, its entry is removed from the list.

As a first step, we designed Algorithm 1, which simply keeps track of the total rate of all flows and assigns each flow a share that is weighted by the flow's priority. Variables are explained in Table 1. Intuitively, it might seem that this simple algorithm would perform well, but our initial tests have shown that it is in fact unsatisfactory. Before we proceed to an improved version of the algorithm, we now illustrate the problem with some of our intermediate results.

Algorithm 1 Active FSE Rate Control

```
Require: $CC\_R$ and $new\_DR$
Ensure: $FSE\_R$
 1: $S\_P \leftarrow 0$
 2: $S\_CR \leftarrow S\_CR + CC\_R - FSE\_R(f)$
 3: for all flows $i$ in FG do
 4:     $S\_P = S\_P + P(i)$
 5: end for
 6: for all flows $i$ in FG do
 7:     $FSE\_R(i) \leftarrow min(new\_DR, ((P(i) * S\_CR)/S\_P))$
 8:     send FSE_R(i) to the flow i
 9: end for
```

Variables	Description
CC_R	The rate received from flow's congestion controller when a flow calls UPDATE
new_DR	The desired rate received a flow when it calls UPDATE
FSE_R	The calculated rate by the FSE
S_CR	The sum of the calculated rates of all flows in the same FG; this value is used to calculate the sending rate
FG	A group of flows having the same FGI, and hence sharing the same bottleneck
P	The priority of a flow which is received from the flow's congestion controller; the FSE uses this variable for calculating FSE_R
S_P	The sum of all the priorities
DELTA	This is used to calculate the difference between CC_R and previously stored FSE_R

Table 1: Names of variables used in algorithms 1 and 2

We implemented the FSE in ns-2 and simulated the behavior of congestion controlled flows using a dumbbell network topology (bottleneck capacity 10 Mbit/s, RTT 10 ms, packet size 1000 bytes, queue length of 13 packets[1]; for simplicity, unless otherwise mentioned, senders always had enough data to send.[2] The current implementation only supports two rate-based protocols: Rate Adaptation Protocol (RAP) [10] (because it is a simple rate-based Additive Increase – Multiplicative Decrease (AIMD) scheme, hence representing a whole class of TCP-like mechanisms) and TCP Friendly Rate Control (TFRC) [5] (because it is the only standardized congestion control mechanism aimed at supporting media flows).

It is clear from the algorithm, and was also confirmed in our simulations, that the FSE achieves precise fairness among the flows. This is important, as it is a requirement for WebRTC [6] – but because coupling congestion controllers should help avoid competition at the bottleneck, we expected reduced queuing delay and packet loss, while achieving at least as much throughput as of a single flow. While the latter requirement was also fulfilled by this algorithm, the results with Algorithm 1 were disappointing regarding queuing delay and packet loss.

The loss ratio and average queue length with FSE-controlled vs. non-FSE-controlled RAP and TFRC flows are illustrated in Figures 1, 2, 3, and 4. Since we only highlight a problem, every data point in these graphs is the result of a single simulation run. It can be seen that, with the FSE, the loss ratio improves as the number of flows grows, but the average queue length is higher. Results were even worse with TFRC.

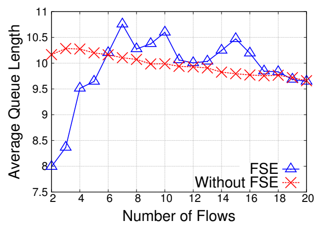

Figure 1: Average queue length (TFRC)

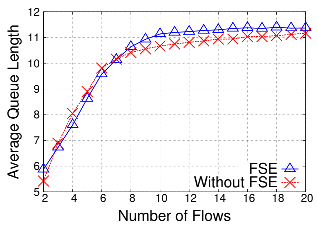

Figure 2: Average queue length (RAP)

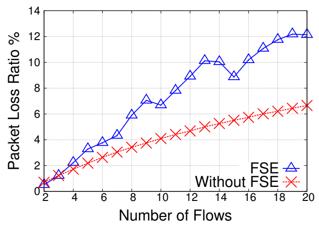

Figure 3: Loss ratio (TFRC)

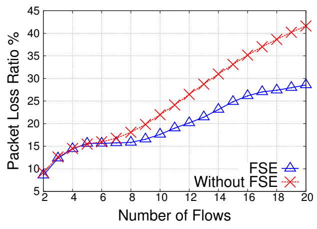

Figure 4: Loss ratio (RAP)

To address these problems, we investigated the queue growth over time with and without the FSE. With the FSE, the queue essentially oscillates between empty and full, but it does not always drain. However, in the same test without the FSE, the queue drains more often (see [7]). This is because the FSE *de-synchronizes* the flows. For example, consider two RAP flows, each sending at a rate X. If one of these flows tries to increase its rate and immediately experiences congestion, it halves its rate, which reduces the aggregate rate from $2X$ to $1.5X$. However, without the FSE, when the two flows get synchronized, both halve their rate when congestion occurs which also halves the rate of the aggregate. Synchronization is usually regarded as a detrimental network effect, but in this case, it appears to play out positively.

In order to fix the loss ratio and average queue growth, we updated our algorithm to emulate a similar behavior by proportionally reducing the aggregate rate on congestion (Algorithm 2). To better emulate the behavior of a single flow, we additionally limited the aggregate rate growth (in the absence of congestion) of N flows to I/N, where I is the flow's increase factor. In order to avoid over-reacting to congestion, we set a timer that prohibits flows other than the flow that just reduced its rate from changing their rate for two RTT periods (of the flow that reduced its rate). We decided to use 2 RTTs so that other flows do not react to the same loss interval. We assume a loss interval to persist for up to one RTT and added another RTT to compensate for fluctuations in the measured RTT value.

A local variable $DELTA$ is used for calculating the difference between CC_R and previously stored FSE_R. When $DELTA$ is negative, we adjust the aggregate and set a timer for 2 RTTs. When the timer is not set or expired, flows operate as before and increase their rates by I/N until congestion is experienced. As we will show in the next section, these

[1]This is based on the bandwidth×delay product (BDP). We repeated our tests with different queue lengths and found no significant differences.

[2]This may not be a totally unreasonable assumption for modern multimedia systems, which may be able to closely track the available bandwidth (cf. [8]). However, the actual behavior is codec-dependent and hard to characterize. At the time of writing, the RMCAT group is working on suitable test cases; in the absence of a solution in this space, we opted to investigate two extreme ends of the spectrum – the case where applications can always send data, and the case where a codec cannot adapt to the available bandwidth at all (Section 4).

changes largely removed the problems that we observed with the first version of our algorithm.

Algorithm 2 Conservative Active FSE Rate Control

```
Require: $CC\_R$, $new\_DR$ and $RTT$
Ensure: $FSE\_R$
 1: $S\_P \leftarrow 0$
 2: if $Timer$ has expired or not set then
 3:     $DELTA \leftarrow CC\_R - FSE\_R(f)$
 4:     if $DELTA < 0$ then ▷ Reduce the sum proportionally
 5:         $S\_CR \leftarrow S\_CR * CC\_R/FSE\_R(f)$
 6:         Set $Timer$ for 2 RTTs
 7:     else
 8:         $S\_CR \leftarrow S\_CR + DELTA$
 9:     end if
10: end if
11: for all flows $i$ in FG do
12:     $S\_P = S\_P + P(i)$
13: end for
14: for all flows $i$ in FG do
15:     $FSE\_R(i) \leftarrow min(new\_DR, ((P(i) * S\_CR)/S\_P))$
16:     send FSE_R(i) to the flow i
17: end for
```

4. EVALUATION

Evaluations were carried out using ns-2 simulations[3] with the same setup as described in the previous section, except that we used a larger RTT of 100ms (and half-BDP queue of 62 packets – we also tested other queue lengths and saw consistently lower queuing delay, see [7]) to better show the delay effect. Different from Section 3, however, all tests reported here were carried out 10 times with different randomly picked start times over the first second. This produced results that had such a small standard deviation (the worst case was 0.2%) that we opted against showing error bars for the sake of clarity.

Figures 5 and 6 illustrate that the updated algorithm achieves a consistent reduction of the average queuing delay both for TFRC and RAP. Figure 8 shows that the loss ratio gain for FSE-controlled RAP flows also becomes noticeable as the number of flows increases. However, the result is less favorable for TFRC, as shown in Figure 7. This is because forcing TFRC to use a lower rate than what its congestion controller has derived causes it to increase its rate more aggressively. From [5], TFRC increases by at most 0.22 packets per RTT, as a result of the deterministic length of loss intervals measured by the receiver. When TFRC uses a lower rate than planned, the loss interval gets artificially prolonged at the receiver, which then calculates a lower value for the loss event ratio p, which in turn provokes a faster rate increase at the sender.

Sending very little obviously produces a small queue and reduces packet loss; however, because Algorithm 2 tries to emulate the behavior of one flow, it should not have a significantly smaller throughput than a single flow. As expected, in all tests, the link utilization with the FSE was at most equal or smaller than without the FSE. However, link utilization of the FSE-controlled RAP flows is *higher* than the link utilization of a single RAP flow. In contrast, for the FSE-controlled TFRC flows, link utilization is in some cases less than the link utilization of one flow, but the difference appears rather marginal (3% less in the worst case in our tests).

[3]source code is available at: `http://safiquli.at.ifi.uio.no/coupled-cc`

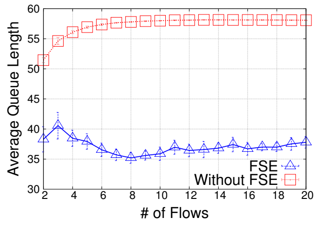

Figure 5: Average queue length (TFRC)

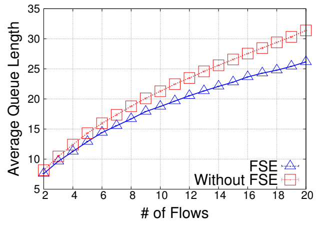

Figure 6: Average queue length (RAP)

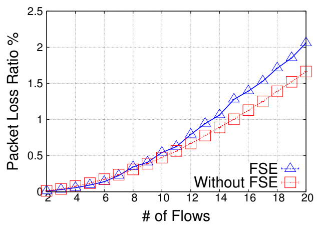

Figure 7: Loss ratio (TFRC)

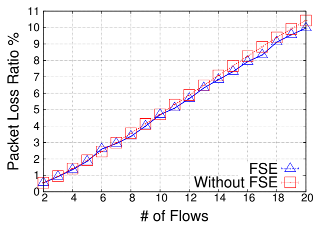

Figure 8: Loss ratio (RAP)

To achieve prioritization, one of the requirements of RMCAT, the FSE can calculate and assign rates based on a priority. Figure 9 shows how two FSE-controlled flows change their rates based on the assigned priorities over time. The two flows started out with a priority of 1 each. After 100 seconds, the priority of flow 1 was decreased to 0.66, 0.42, 0.25 and 0.11 after 100, 150, 200 and 250 seconds, respectively. This means that a high priority flow can easily get the desired rate from the FSE without requiring any further changes in its congestion controller. The first 100 seconds of this graph also illustrate the perfect fairness that is enforced by our algorithm; we do not show Jain's fairness index because the result was always 1 in our tests.

We also investigated the fairness of 2-5 RAP and TFRC flows with different RTTs between them, with ratios up to 48:24:12:6:3. While the FSE enforces perfect fairness irrespective of the RTT, the fairness without the FSE degrades heavily in some cases (for details, please refer to [7]).

RMCAT targets interactive media flows, with a focus on video and audio. Other than the bulk data transfers that we have used in our evaluation so far, such flows do not always keep the send buffer full. Using such "greedy" traf-

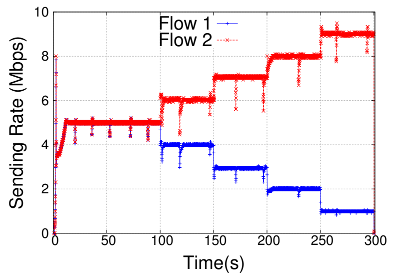

Figure 9: Flow 1 changing its priority coupled via the FSE

fic is a reasonable starting point because a mechanism that fails when its send buffer is constantly full has little chance of success when the buffer occasionally runs empty. There is an ongoing discussion in RMCAT on how to best evaluate congestion control mechanisms, given the multitude of available codecs and their different behaviors; but there is some consensus that modern codecs are able to track the transport's calculated rate quite precisely.

In the face of these complications, we decided to use a simple approach to evaluate how well our mechanism would work with media traffic. 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 therefore ran a simulation with two flows: an application limited flow, sending based on a video trace, and a greedy flow. As it can be observed from Figure 10, in the presence of the congestion, FSE-controlled flows proportionally reduce their rates together, whereas synchronization causes the application-limited flows to over-react without the FSE (e.g., in the congestion events at t=5, 10 and 20 seconds in Figure 11).

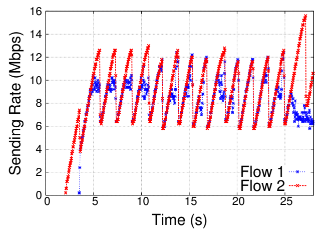

Figure 10: Application limited flow and greedy flow – with FSE

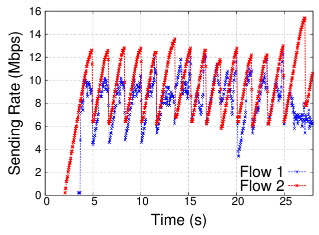

Figure 11: Application limited flow and greedy flow – without FSE

Figure 12 illustrates the behavior of a greedy flow with low priority (0.2) and an application limited flow with a higher priority (1) that is sending based on a video trace. It can be observed that the low-priority flow can grab unused bandwidth as long as there is enough capacity. The bandwidth was not completely utilized in these tests because the simulation time was based on the total duration of the video trace, which was too short for the low priority flow to reach the capacity limit.

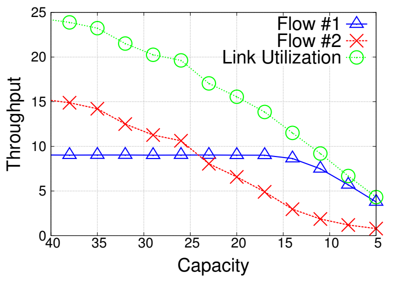

Figure 12: High-priority (1) application-limited flow #1 is hardly affected by a low-priority (0.2) greedy flow #2 as long as there is enough capacity for flow #1.

We conducted a series of simulations using synthetic background traffic in order to emulate a situation that is typical for the Internet. For this purpose we used TMIX [13], which is a tool to generate realistic TCP application workload in

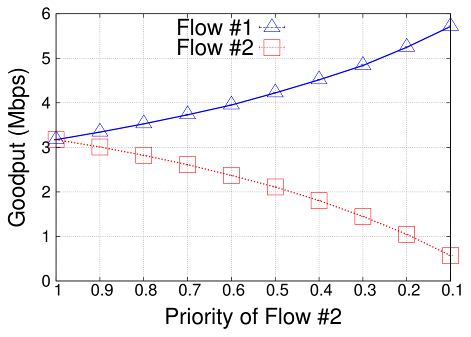

Figure 13: Goodput of two FSE-controlled flows competing with synthetic traffic

ns-2. The traffic used in our simulation is taken from 60-minute trace of campus traffic at the University of North Carolina, which is available from the common TCP evaluation suite [2].

We employed a pre-processed version of this traffic which is adapted to provide an approximate load of 50% on a 10 Mbps bottleneck link based on the network topology discussed in previous sections over the course of 300 sec as simulation time. The pre-processing also included the removal of non-stationarity in the background traffic pattern by randomly shuffling different portions of the traffic pattern. The RTT of background TCP flows generated by TMIX fluctuates between the range of 80∼100 ms while the RTT of foreground TFRC flows was statically set to 100 ms, and foreground and background traffic shared the bottleneck queue.

Figure 13 shows the goodput values of two TFRC flows with FSE in the presence of background synthetic traffic when the priority of the first flow is set to 1, while the other flows' priority is varied. As it can be seen from the graph, the goodputs of flows 1 and 2 are very close to the theoretical value that one might expect: for example, when the priority of flow 2 is 0.2, 0.5 and 0.8, the goodput ratio is 0.199 (instead of 0.2), 0.499 (instead of 0.5) and 0.799 (instead of 0.8), respectively. These are surprisingly precise values, seen by the receivers in the presence of synthetic background traffic with various numbers of arriving and departing flows and RTTs at any instance of time.

5. CONCLUSIONS

We have presented the coupled congestion control mechanism that is currently being proposed for WebRTC in the IETF RMCAT group. Simulations with the two congestion control mechanisms RAP and TFRC indicate that, our method not only satisfies the requirements of controllable fairness with prioritization, but, by emulating the behavior of a single flow, also reduces queuing delay and packet loss without significantly affecting throughput. In case of RAP, we even saw these effects combined with *better* link utilization than with a single flow. The difference in behavior between the two mechanisms highlights the need to evaluate our scheme with each mechanism it is applied to.

We plan to test our method in real life as a next step. The congestion control of RMCAT is currently under development, and will probably be delay based; we therefore need to test our scheme with a delay-based congestion control too. To incorporate WebRTC's data channel, we will investigate coupling with window-based protocols too; then we can control TCP like E-TCP and EFCM, which will enable us to compare the mechanisms against each other. At this point, it will also be necessary to investigate the effect of coupling different congestion controllers together. The evaluations in this paper were also limited to a scenario where SBD is based on multiplexing, not measurements. With measurement-based SBD, flows between different host pairs can be controlled, which means that the flows will also have different RTTs – another factor that needs to be incorporated in future evaluations.

6. ACKNOWLEDGMENTS

This work is partially supported by the European Union through the FP7-ICT project RITE under contract number 317700. We would like to thank Dr. David Hayes for his guidance in conducting TMIX tests.

7. REFERENCES

[1] SPDY: An experimental protocol for a faster web. `http://www.chromium.org/spdy/spdy-whitepaper`. Last Accessed:06/07/2013.

[2] L. Andrew, S. Floyd, and G. Wang. Common TCP evaluation suite. http://tools.ietf.org/id/draft-irtf-tmrg-tests-02.txt, 2009.

[3] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion manager architecture for internet hosts. In *Proc. ACM SIGCOMM*, 1999.

[4] L. Eggert, J. Heidemann, and J. Touch. Effects of ensemble TCP. *USC/Information Sciences Institute*, 7(1), December 1999.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *ACM SIGCOMM*, 2000.

[6] C. Holmberg, S. Hakansson, and G. Eriksson. Web real-time communication use-cases and requirements. Internet-draft draft-ietf-rtcweb-use-cases-and-requirements-12.txt (work in progress), 2013.

[7] S. Islam, M. Welzl, S. Gjessing, and N. Khademi. Coupled congestion control for RTP media. Technical Report 440, March 2014. `https://www.duo.uio.no/handle/10852/39177`.

[8] M. Nagy, V. Singh, J. Ott, and L. Eggert. Congestion control using fec for conversational multimedia communication. *arXiv preprint arXiv:1310.1582*, 2013.

[9] P. Natarajan, P. D. Amer, and R. Stewart. Multistreamed web transport for developing regions. In *SIGCOMM NSDR workshop*, 2008.

[10] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *IEEE INFOCOM*, 1999.

[11] M. Savorić, H. Karl, M. Schläger, T. Poschwatta, and A. Wolisz. Analysis and performance evaluation of the EFCM common congestion controller for TCP connections. *Computer Networks*, 49(2):269–294, 2005.

[12] J. Touch. TCP control block interdependence. RFC 2140, April 1997.

[13] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith. Tmix: A tool for generating realistic TCP application workloads in ns-2. *SIGCOMM Comput. Commun. Rev.*, 36(3):65–76, July 2006.

[14] M. Welzl, S. Islam, and S. Gjessing. Coupled congestion control for RTP media. Internet-draft draft-welzl-rmcat-coupled-cc-02 (work in progress), 2013.

[15] M. Welzl, F. Niederbacher, and S. Gjessing. Beneficial transparent deployment of SCTP: the missing pieces. In *IEEE GLOBECOM*, 2011.