Managing Real-Time Media Flows through a Flow State Exchange

Safiqul Islam, Michael Welzl, David Hayes and Stein Gjessing
Networks and Distributed Systems Group, Department of Informatics
University of Oslo, Norway
Email: {safiquli, michawe, davidhay, steing}@ifi.uio.no

Abstract—When multiple congestion controlled flows traverse the same network path, their resulting rate is usually an outcome of their competition at the bottleneck. The WebRTC / RTCWeb suite of standards for inter-browser communication is required to allow prioritization. This is addressed by our previously presented mechanism for coupled congestion control, called the Flow State Exchange (FSE). Here, we present our first simulation results using two mechanisms that have been proposed for IETF standardization: Google Congestion Control (GCC) and Network-Assisted Dynamic Adaptation (NADA). These two mechanisms exhibit aspects that allow us to use a simpler “passive” algorithm in our FSE. Passive coupling allows a less time-constrained request-response style of signaling between congestion control mechanisms and the FSE, which enables the FSE to run as a stand-alone management tool.

I. INTRODUCTION

Despite the fact that video conference applications have been widely used for many years, except for TCP Friendly Rate Control (TFRC) [1], there is no standardized congestion control mechanism available. Therefore, the “RTP Media Congestion Avoidance Techniques” (RMCAT)1 IETF Working Group has been established to develop standards for RTP-based interactive real-time media, with a focus on helping WebRTC (and the related IETF set of standards, RTCWeb). WebRTC enables interactive real-time communication between web browsers, facilitating a range of applications such as seamless video conferencing, telephony and interactive gaming. These should all be accessible as part of the web surfing experience and not require the installation of additional software. One important RTCWeb requirement for RMCAT standards is the ability to allow the WebRTC application programmer to assign priorities to flows. These priorities control how the available capacity is shared [2].

In a recent work [3] we proposed the Flow State Exchange (FSE); a mechanism that couples congestion control for RMCAT. The FSE couples the congestion controls of competing RMCAT flows with priorities—even between combinations of different congestion control mechanisms. It enables the sender to precisely control prioritized bandwidth sharing, removing self-competition. This improves the overall performance, reducing delay and loss [3].

Having shown in [3] how the FSE performs with the Rate Adaption Protocol (RAP) [4] and TFRC [1], we now investigate its usage for two proposed RMCAT congestion control mechanisms: Network-Assisted Dynamic Adaptation (NADA) [5] and Google Congestion Control (GCC) [6]. Among these congestion control mechanisms, GCC has already been deployed in Google Chrome, Chromium, Firefox, and Opera browsers.

Applying the FSE to different congestion control algorithms requires a small adaption to the FSE algorithm. In this paper, we show how this is done for NADA and GCC, leading us to an interesting observation: because both NADA and GCC update their rates at fixed time intervals – and not as a function of the RTT – we can use an even simpler, “passive” version of our algorithm. Different from our previous “active” version [3], which required immediate callbacks, the passive FSE algorithm can be implemented as a simple request-response type server, with less signaling overhead and relaxed time constraints. This makes it possible to run the FSE as a stand-alone application, potentially turning it from an integral part of an application’s congestion control into a separate tool for managing priorities between flows.

We believe that a standalone FSE tool will have uses beyond the scope of just WebRTC, where support of priorities is a requirement [2]. This belief is supported by the results of a survey in which we asked 139 students and work colleagues whether they had experienced network traffic from different applications interfering with each other on their computers. 96 of them responded “yes, and I found it annoying”, 19 chose “yes but I did not care” and the remaining 24 chose “No: this never happened – or if it did, I did not notice”. Asked if they would use a tool that would be easy to handle and would let them prioritize how applications access the network, 89 participants said yes, 11 said no, and 39 used a free-text field to give a different answer, such as (relatively common) “I already use QoS mechanisms”. While the QoS argument is valid, it requires knowledge of and access to the most common bottleneck link (often the access point), that not all users have.

This paper is organized as follows: Section II presents background information (NADA and GCC) and related work. Section III explains our FSE algorithm and how we have changed it for NADA and GCC. In Section IV we show some evaluation results using ns-2 simulations for NADA and simulations using the Chromium browser for GCC, with

1http://tools.ietf.org/wg/rmcat/

978-1-5090-0223-8/16/$31.00 © 2016 IEEE
conclusions in Section V.

II. BACKGROUND AND RELATED WORK

Network-Assisted Dynamic Adaptation (NADA) [5, 7] is a congestion control scheme designed for interactive real-time media applications. In NADA, the receiver combines both implicit (per-packet drops and one way delay measurements) and explicit signals into a composite congestion signal, and periodically reports back to the sender using the Real-Time Transport Control Protocol (RTCP). The sender thus regulates its sending rate upon receipt of the RTCP feedback. It can be seen from the simulation results in [7] and [8] that the algorithm achieves good fairness by maintaining a stable queue when multiple NADA and TCP flows compete.

Google Congestion Control (GCC) [6] is another congestion control algorithm proposed for WebRTC. GCC employs two controllers: (i) a sender side controller, and (ii) a receiver side controller. The sender side controller controls the bandwidth based on packet loss, and the receiver side controller controls the bandwidth based on delay. These two controllers are designed to increase the rate in the absence of congestion. Interestingly, the sender side controller only reacts to losses over 10 percent. In [9, 10], the authors showed that in a previous version, GCC starved when competing with TCP. However, recent changes [11] show that the algorithm now can achieve good fairness while minimizing queuing delay.

There have been a number of earlier end system flow management efforts. The first work to combine flow management used a mechanism that enabled TCP’s Control Block (TCB) to be shared among flows [12]. This shared state is used to initialize new connections. Ensemble TCP (E-TCP) builds upon [12], extending it to let multiple parallel flows immediately benefit from each other [13]. The authors of [14] extend E-TCP’s concepts with their Ensemble Flow Congestion Manager (EFCM). In E-TCP, the ensemble of $n$ TCP connections is no more aggressive than one TCP connection, however, EFCM allows the ensemble to be as aggressive as $n$ separately controlled TCP connections. In [15], an integrated Congestion Manager (CM) was proposed that replaces each flow’s congestion control mechanism with its own common rate based mechanism. CM was eventually standardized [16], however, it is hard to implement and has never been widely deployed. Our FSE does not provide a new congestion control mechanism, rather it provides a simple protocol interface that facilitates the coupling of flows using their own congestion controllers. This enables it to support an extensible list of transport protocols without the implementation complexity or single protocol dependency that has hampered adoption of the other mechanisms.

III. THE FLOW STATE EXCHANGE (FSE)

Because prior approaches such as the Congestion Manager did not see wide-spread deployment and have been reputed to be too complex to implement, we have opted for an approach that minimizes changes to existing congestion control mechanisms [17], called the “Flow State Exchange” (FSE).
Fig. 2: Active and Passive versions of the FSE. CC_R is the rate received from the flow’s congestion controller. FSE_R(f) is the rate calculated by the FSE. Variables are explained in Section III-E and Table I.

Also delay the feedback from the FSE. This prevents the FSE from imposing strict fairness (in the test shown in Fig. 3, both flows have the same priority).

Consider two flows, flow 1 with a slow update frequency (i.e., a long RTT) and flow 2 with fast update frequency (short RTT). If flow 1 sends at a high rate and flow 2 wants to decrease the rate because it sees congestion, a passive FSE algorithm will make an overall decision for the rate aggregate and assign an appropriate rate to each flow (this is a necessary part of any FSE algorithm that assigns priorities). This means that it will probably record that flow 1 should reduce its rate. However, flow 1 does not incorporate this update for a long time and keeps on sending too fast. Thus, if the rate update of flow 2 is based on an assumption about flow 1, this assumption is wrong and the overall outcome is undesirable leading to the behavior we observe in Fig. 3.

This can be solved by choosing the active variant (see Fig. 2(a)), where all flows immediately incorporate updates from each other (as in [3]). As our discussion of NADA and GCC will show that the active version is superfluous to their needs.

B. Increasing and decreasing rates

If the FSE algorithm leaves the increase behavior unchanged, the overall increase behavior is more aggressive when compared to a single flow. This is the behavior adopted in [17, 3]. An FSE algorithm can also force the increase behavior to be exactly like one flow. For example, if the increase behavior is additive as in AIMD, this can be done by only allowing one flow to increase the rate of the aggregate per update interval. This would be a more conservative behavior.

If the FSE algorithm leaves the decrease behavior unchanged, the overall behavior is more aggressive than only a single flow. Consider, for example, 10 AIMD (Additive-Increase, Multiplicative-Decrease) flows sending with a rate of \( x \) bits per second each without using an FSE. The aggregate rate of these flows is \( 10x \). If only one of them experiences congestion and that causes that flow to halve its rate (MD with multiplication factor 0.5), the rate aggregate will only be reduced to \( 9.5x \). In reality, flows often get synchronized, increasing the chance for multiple flows to see congestion during the same round-trip time. If all flows saw congestion at the same time in our example, the aggregate would end up at a rate of \( 5x \). Thus, depending on how synchronized the flows are, the outcome is somewhere between halving the entire aggregate or only halving the rate of a single flow.

The “active conservative” algorithm in [17, 3] forces the decrease behavior to be like one flow. This decision was taken to allow the queue to drain more often because the goal was to achieve lower loss and delay. A single congestion control like TCP also tries to avoid reacting more than once per loss event (RTT), so this behaviour also had to be incorporated into the design.

C. Application to NADA

NADA has the following relevant properties:

1) Building a stable queue: NADA primarily reacts to delay signals and updates rates gradually when a receiver reports a standing, increasing or decreasing queue; Fig. 4 illustrates...
Fig. 4: One-way delay of one NADA flow and one GCC flow across a 1 Mbps, 50 ms base delay link (separate simulations).

the delay characteristics of one NADA flow compared with that of a GCC flow.

2) Accelerated ramp up: A NADA sender uses two different modes to update its sending rate: gradual rate update and accelerated ramp up. In accelerated ramp up mode, a NADA flow increases its rate faster when the reported queuing delay is close to zero [5].

3) Rate update frequency: A NADA receiver sends an update to the sender every 100 ms, and the sender updates its sending rate whenever feedback arrives.

Derived implications for the FSE algorithm:

a) Increase: There is no obvious reason to change the increase behavior of NADA, so the FSE can leave it unaltered, similar to the version in [3].

b) Decrease: Here, the active conservative algorithm proposed in [3] drains the queue more often than multiple NADA flows normally would do because it proportionally reduces the rates of all flows. While this by itself is not bad, it triggers the accelerated ramp up more often, in turn rendering the flows more aggressive again. Hence we found that leaving the decrease behavior unchanged is a better choice.

c) Active or passive: Because the rate update frequency is fixed, any error introduced by passive updates persists for at most the length of the update interval. Hence, it is limited by a fixed value (as opposed to the RTT, e.g. in TCP), and hence the predictable regular updates make the “active” FSE behavior superfluous.

D. Application to GCC

The following two properties of GCC are relevant for the FSE:

1) Builds a stable queue: Since GCC reacts to delay signals, it builds a stable queue similar to NADA. Fig. 4 illustrates the delay characteristics of one GCC flow.

2a) Rate update frequency: From the most recent specification of GCC in [6], the frequency at which a sender updates its rate (immediately upon receiving feedback from the receiver or based on a timer) is not fully defined. According to [9], the frequency of sending feedback from the receiver is still an open issue. The Google implementation in the version of Chromium that was used for this paper (47.0.2494.0) updates the sending rate every 25 ms and whenever a feedback message arrives; these messages were sent every 50 ms second or if there is rate drop of at least 3%. The latter rarely occurred in our tests, meaning that approximately every other rate update was caused by the 25 ms timer.

Derived implications for the FSE algorithm:

b) Increase: There is no obvious reason to change the increase behavior of GCC, so the FSE can leave it unaltered, similar to the version in [3].

c) Decrease: In early tests, we did not see any major issues with the decrease behavior of the active conservative algorithm proposed in [3]. However, given that the mechanism is by itself delay based, the benefits of such a conservative decrease rule are limited so we opted for leaving the decrease behavior unchanged for the sake of simplicity.

d) Active or passive: Because rate update timers operate independently from the RTT, the frequency at which GCC flows change their rates and hence access the FSE is also RTT-independent. Again, the “active” FSE behavior is probably unnecessary.

E. A simple passive FSE algorithm for NADA and GCC

The outcome of the active FSE algorithm design in [3] is that flows using the FSE are a bit more aggressive than one flow but less aggressive than multiple: they increase the aggregate rate faster but allow the queue to drain more often. As our discussion has shown, many of the design decisions taken in our active variants are either unnecessary or inappropriate for NADA or GCC, rendering the resulting algorithm significantly simpler.

The passive FSE algorithm is summarized in Algorithm 1; variables are explained in Table I. The FSE contains a list of all flows that have registered with it. Each flow has the following state:

(i) a unique flow number to identify the flow,
(ii) the Flow Group (FG) identifier indicating the group it belongs to (all flows in the same FG are assumed to share the same bottleneck in the network),
(iii) a flow priority P,
(iv) and the calculated rate FSE_R.

Each FG contains one global variable S_CR which is the sum of the calculated rates of all flows in the same FG. The FSE keeps track of the total rate of all flows and assigns each flow a share that is weighted by the flow’s priority.
In this paper, we first show the prioritization results for both NADA and GCC flows, then the results using two pertinent test cases are: a) "media pause and resume" test case\(^4\) and b) "round-trip time fairness" for five RMCAT flows with different round-trip times. The one way propagation delays of flows are 10 ms, 25 ms, 50 ms, 100 ms, and 150 ms, respectively. Figs. 7(b) and 8(b) show that the FSE helps both NADA and GCC flows to converge more quickly than without the FSE (Figs. 7(a) and 8(a)).

Figs. 9 and 10 show results for both NADA and GCC with two continuous and one intermittent RMCAT flows in the “media pause and resume” test case\(^2\). Figs. 9(a) and 10(a) show the rate and delay characteristics without the FSE, and Figs. 9(b) and 10(b) show results with the FSE for NADA and GCC respectively. In this test, all three flows start with the same priority. At around 50 s, the priorities of streams 2 and 3 were decreased to 0.66 and 0.33, respectively. This means that flows get their assigned proportion of the available capacity without requiring any further changes in the congestion controllers.

### IV. Evaluation

We implemented the FSE in ns-2\(^2\) and the Chromium browser\(^3\). The actual behavior of media flows in the browser is codec-dependent and hard to characterize. The IETF RMCAT group has prepared a number of test cases\(^2\) to evaluate the performance of congestion control mechanisms. In accordance with\(^2\), all tests use the dumbbell topology depicted in Fig. 5, with a bottleneck queue length chosen to produce a maximum delay of 300 ms. For all figures except Figs. 7 and 8, the one-way path delay is 50 ms. From Fig. 7 to Fig. 8 the link capacity is 4 Mbps, and from Fig. 9 to Fig. 10 the link capacity is 3.5 Mbps.

In this paper, we first show the prioritization results for both NADA and GCC flows, then the results using two pertinent test cases\(^4\) from\(^2\) in order to document the efficacy of our proposed solution, and finally test our system’s efficacy when feedback is delayed. The two test cases are:

1. **Round-trip time fairness**: In this test case, five media sources S1, S2, S3, S4, and S5 are connected to D1, D2, D3, D4, and D5 media sinks, respectively (n=5 in Fig. 5). The one way base delays are, 10 ms for S1-D1, 25 ms for S2-D2, 50 ms for S3-D3, 100 ms for S4-D4, and 150 ms for S5-D5, respectively.

2. **Media pause and resume**: In this test case, three media sources S1, S2, and S3 are connected to D1, D2, and D3, respectively (n=3 in Fig. 5). S2 is paused for 20 seconds at around 40 seconds.

### Algorithm 1 Passive FSE Rate Control for flow f

**Require:** CC\(_R\)(f)

**Ensure:** FSE\(_R\)(f)

```
1: S_P ← 0
2: S_Cr ← S_Cr + CC\(_R\)(f) - FSE\(_R\)(f)
3: for all flows i in FG do
   4:   S_P ← S_P + P(i)
5:   end for
6: FSE\(_R\)(f) ← ((P(f) × S_Cr) / S_P))
7: send FSE\(_R\)(f) to the flow
```

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC(_R)(f)</td>
<td>The rate received from f’s congestion controller when a flow calls UPDATE</td>
</tr>
<tr>
<td>FSE(_R)(f)</td>
<td>The rate calculated by the FSE for flow f</td>
</tr>
<tr>
<td>S_Cr</td>
<td>The sum of the calculated rates of all flows in the same FG; this value is used to calculate the sending rate</td>
</tr>
<tr>
<td>FG</td>
<td>A group of flows having the same FGI, and hence sharing the same bottleneck</td>
</tr>
<tr>
<td>P(i)</td>
<td>The priority of flow f which is received from the flow’s congestion controller; the FSE uses this variable for calculating FSE(_R)(f)</td>
</tr>
<tr>
<td>S_P</td>
<td>The sum of all the priorities</td>
</tr>
</tbody>
</table>

### TABLE I: Names of variables used in Algorithm 1

Results of tests from\(^2\) are commonly presented in the form of the sending rate and packet transit delay evolution over time. This illustrates the dynamic behavior and we follow the same format in this paper.

#### A. Prioritization results

The FSE calculates and assigns rates based on priorities. Fig. 6 shows how three FSE-controlled flows change their rates based on the assigned priorities over time. Figs. 6(a) and 6(b) illustrate the sending rates for three FSE-controlled NADA flows and three FSE-controlled GCC flows, respectively. The three flows started out with a priority of 1 each. After 50 s, the priorities of streams 2 and 3 were decreased to 0.66 and 0.33, respectively. This means that flows get their assigned proportion of the available capacity without requiring any further changes in the congestion controllers.

#### B. Test case results

Figs. 7 and 8 show results for the test case “round-trip time fairness” for five RMCAT flows with different round-trip times. The one way propagation delays of flows are 10 ms, 25 ms, 50 ms, 100 ms, and 150 ms, respectively. Figs. 7(b) and 8(b) show that the FSE helps both NADA and GCC flows to converge more quickly than without the FSE (Figs. 7(a) and 8(a)).

Figs. 9 and 10 show results for both NADA and GCC with two continuous and one intermittent RMCAT flows in the “media pause and resume” test case\(^2\). Figs. 9(a) and 10(a) show the rate and delay characteristics without the FSE, and Figs. 9(b) and 10(b) show results with the FSE for NADA and GCC respectively. In this test, all three flows start with the same priority. At around 40 s flow 1 was paused for 20 seconds. Figs. 9(b) and 10(b) show that the FSE distributes the aggregate fairly by enforcing strict fairness. The FSE also improves the delay spike introduced by a NADA flow (see Fig. 9) or a GCC flow (see Fig. 10) after a long pause.

#### C. Delayed feedback tests

In order to see if our mechanism would be robust against operating system disturbance if the FSE would be run as a stand-alone application, we ran tests with 2 GCC flows where we delayed the signal between the congestion controller

---

\(^2\)http://www.isi.edu/nsnam/ns/
\(^3\)https://www.chromium.org
\(^4\)Test results of all the test cases of the active version have been presented, see [22, 23].
Fig. 6: Sending rates of 3 NADA flows and 3 GCC flows as the priorities of flows are varied at around 50 seconds. (Note that markers identify the line and not plotted points)

Fig. 7: Sending rates and delays of five NADA flows with one way delays of 10 ms, 25 ms, 50 ms, 100 ms, and 150 ms. The FSE not only enforces perfect fairness but also helps the congestion control mechanism to converge quickly. Delay is largely unaffected. (Note that markers identify the line and not plotted points)

We also carried out tests with delay values beyond the ones shown in Fig. 11 (even up to 500 ms), with similar results. Less time-constrained request-response style signaling between congestion control mechanisms and the FSE adds to its robustness. This makes it possible to run the FSE as a stand-alone application.
Fig. 8: Sending rates and delays of five GCC flows with one way delays of 10 ms, 25 ms, 50 ms, 100 ms, and 150 ms. The FSE not only enforces perfect fairness but also helps the congestion control mechanism to converge quickly. Delay is largely unaffected. (Note that markers identify the line and not plotted points)

Fig. 9: Sending rates and delays of two continuous and one intermittent NADA flows, with and without the FSE. (Note that markers identify the line and not plotted points)

V. CONCLUSIONS

In this paper, we have evaluated our FSE management mechanism with the two congestion control algorithms NADA and GCC. The evaluations show that the simpler, passive version of FSE works very well with both NADA and GCC flows. Both NADA and GCC update their rates at fixed intervals, and we have argued that this is the reason the passive version of the FSE manages these flows so well.

A passive FSE uses less signaling than an active one. In addition we have shown that a passive FSE works well with relaxed timing constraints vis-à-vis the congestion controllers, making it possible to run the FSE as a stand-alone tool and even on a separate system, i.e. as a server that clients query to obtain the right sending rate. While this could multiply the benefits of the FSE by uniformly controlling several hosts, the FSE always needs to be aware of the common bottleneck, which may complicate the design of such a system. We plan to investigate this in future work. In the RMCAT context, we will also evaluate the passive FSE with the third currently proposed mechanism, SCReAM [24]. We expect similar results because SCReAM also employs a fixed feedback frequency. However, the passive FSE mechanism could be applied to any set of
flows that share similar properties, not just real-time media flows. We therefore also plan to investigate whether the passive FSE can be used to control such non-real-time media mechanisms in future work.

VI. Acknowledgments

This work is partially supported by the European Union through the FP7-ICT project RITE under contract number 317700. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the overall quality of the paper. The authors would also like to thank Xiaoqing Zhu and Stefan Holmer for helping out with NADA and GCC respectively, and Kristian Hiorth for his advice regarding Chromium.

REFERENCES


