Awesome title here

Managing Real- Time Video and Data Flows with Coupled Congestion Control Mechanism

Tobias Fladby

Thesis submitted for the degree of
Master in programming and system architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

Awesome title here

Managing Real- Time Video and Data Flows with Coupled Congestion Control Mechanism

Tobias Fladby

Awesome title here

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Contents

List of Figures

List of Tables

Preface

Chapter 1

Introduction

1.1 Problem statement

1.2 Contributions

1.3 Research questions

Overall:

- Can two heterogenous control mechanisms be coupled? Will it improve overall performance?

Simplicity:

- Can such a mechanism be designed simple enough for widespread implementation? Moreover can it be easily integrated with other congestion control mechanisms?

Fairness:

- Will both flows get their allocated share of bandwidth when needed?
- Will the coupled flows be fair to other flows sharing the same bottleneck?
- Will the bandwidth be shared according to configured priority?

Delay:

- Can it reduce delay spikes?

Link utilization:

- Will link utilization be equal to a single flow using the full link?

Responsiveness:

- Will any of the congestion control mechanisms be more responsive to congestion in the network?

Packet loss:

- Will any of the flows experience less packet loss?
- Can it reduce packet loss spikes for the flows?

1.4 Organization

Chapter 2

Background

2.1 WebRTC architecture

2.1.1 Real- time communication

2.1.2 Standardization

2.1.3 Protocol Stack

2.1.4 User API

Services

RTCPeerConnection API

DataChannel API

2.1.5 Signalling

NAT

ICE Framework

TURN

STUN

SDP

2.1.6 Encryption

TLS

DTLS

2.1.7 Usage

2.1.8 Browser engine

Aquiring WebRTC statistics

2.2 Transport protocols

2.2.1 TCP

2.2.2 UDP

Message- oriented protocols

2.2.3 RTP and RTCP

2.2.4 SCTP

SCTP offers a point- to- point connection- oriented reliable delivery service while also using the same flow and congestion control algorithms as TCP. As opposed to TCP, SCTP is message- oriented. An SCTP connection is called an association.

SCTP separates application data into chunks, each identified by a separate chunk header. These chunks are bundled into a single SCTP message that consists of an SCTP message header followed by several data chunks. A key feature here is that the data chunks are independently identified with a separate header, thus a single SCTP message can contain data from separate streams of application data. For example one stream being text messages and another being the transfer of a file in a messaging application. The advantage of this packet structure is that it means SCTP can support multi- streaming since it can send multiple data streams in parallel through a single SCTP association.

Multi- streaming means that an application can transmit several independent streams of data in parallel.

SCTP separates application data into chunks, each identified by a separate header. A single SCTP packet can contain several data chunks from different application data streams.

2.3 Congestion control

2.3.1 Loss- based congestion control

2.3.2 Delay- based congestion control

2.4 WebRTC Congestion controls

Video data by nature is large in size so transmitting it creates a lot of traffic. This makes real- time communication challenging because it requires low latency in order to assure a good user experience.

History and previous research [cite relevant stuff, like congestion collapse]has shown that protocols should employ mechanisms that limit the amount of data sent per second to a reasonable level in order to avoid congestion as well as keep the latency low.

2.4.1 Google Congestion Control

RTP by itself only provides simple end- to- end delivery services for multimedia[cite RTP standard], since real- time communication requires congestion control it must implemented on top of RTP. Chromium's WebRTC implementation uses an algorithm called Google Congestion Control [2] to provide the mechanism. It consists of two controllers, one loss- based and one delay- based. The loss- based controller located on the sender- side, uses loss rate, RTT and REMB[Cite REMB message definition] messages to compute a target sending bitrate. The delay- based controller can either be implemented on the receiver- side or sender- side. It uses packet arrival info to compute a maximum bitrate which is passed to the

loss- based controller. The actual sending rate is set to the minimum of the two bitrates.

The loss- based controller

The loss- based controller is run every time a feedback message from the receiver- side is received. If more than 10% of packets have been lost when feedback is received the controller decreases the estimate. If less than 2% is lost it will increase the estimate under the presumption that there is more bandwidth to utilize. Otherwise the estimate stays the same.

The delay- based controller

The delay- based controller consists of several parts: pre- filtering, an arrival- time filter, an over- use detector and a rate controller.

Pre- filtering is used to make sure that channel outages, events unrelated to congestion are not interpreted as congestion. Packets will naturally be delayed when a channel outage occurs so without this filter the algorithm would unnecessarily lower bitrate, thus lowering the quality of the communication for no reason. A channel outage will cause the packets to be queued in network buffers, thus when the channel is restored the packets will arrive in bursts. The filter utilizes the fact that the packet groups will arrive in bursts during a channel outage and merges them under such conditions.

The arrival- time filter is responsible for calculating the queueing time variation which is an estimation of how the delay is developing at a certain time. The goal of the over- use detector is to produce a signal that drives the state of the remote rate controller. The goal of the over- use detector is to compare the queueing time variation obtained as output from the arrival-time filter with a threshold. If the estimate is above the threshold for a certain amount of time and not sinking it wil signal the rate contol.

Performance

2.4.2 NADA

Overview

Receiver agent

Sender agent

2.4.3 SCReAM

2.5 Coupled Congestion Control

2.5.1 Problems with combined controls

The problem There are inherent differences in how quickly different types of congestion control react to congestion and combining them can therefore easily lead to unintentional side- effects. This is a known issue

especially when it comes to combining loss- based controls with delay-based controls. It has been shown to lead to a competition between the flows resulting in spikes in queueing delay and packet loss. The main issue stems from the fact that delay always happens earlier than loss. The reason is that the first thing that happens when there is congestions is that the bottleneck queues will start filling, thus making packet more delayed, but not necessarily dropped until the queue is full or close to full. Intuitively, this means that delay is observable earlier than loss when there is congestion. The consequence of all this is that the delay-based control will decrease it's sending rate sooner than the loss- based will. Such behaviour leads to a very bad dynamic where the delay- based control lowers the bitrate at such an early stage of congestion that the loss-based never experiences packet loss and thus keeps increasing its send rate. The final result is that the delay- based control will get a smaller and smaller share of the available bandwidth because the loss- based control keeps increasing while the delay- based keeps backing down because of the congestion caused by the still- increasing traffic from the loss- based controller.

Coupling the controls A possible solution is to have the controllers cooperate by sharing their information, given that they are both located on the same host and are travelling the same path across the network i.e. coupling the flows. Firstly, such a mechanism could benefit both controllers by giving them more information, and different types of information to base their decisions on. Secondly this could be used to ensure that the loss- based algorithm is fair to the delay- based. One could for instance make sure that the loss- based control also decreases when the delay- based controller decreases the send rate. One mechanism for coupling is "The Congestion Manager" (CM) [1]. CM couples flows by offering a single congestion controller instead. The downside is that it is considered quite hard to implement because it requires an extra congestion controller and strips away all per- connection congestion control functionality, which is a drastic change. A newer solution called "Coupled Congestion Control" [4] combines congestion controls travelling over the same bottleneck while at the same time aiming at being easier to implement than the congestion manager. As opposed to The Congestion Manager, Coupled Congestion Control tries to utilize the congestion control algorithms of the coupled flows by sharing the information they gather among them instead of completely removing them. The mechanism has aready shown promise in [3, 5] when implemented with homogenous congestion controls but has not been tested on heterogenous congestion controls.

Coupled Congestion Control Architecture The design philosophy of Coupled Congestion Control is that the amount of required changes to existing applications should be minimal. The system consists of three elements, Shared Bottleneck Detection(SBD), Flow State Exchange(FSE) and the flows.

Managing flows When a flow starts it will register itself with the FSE and SBD, when it stops it will deregister from the FSE and carry out an UPDATE function call every time their congestion controller calculates a new rate. When a flow registers itself the SBD will assign it to a Flow Group by giving it a Flow Group Identifier. A flow group is defined as a group of flows that share the same bottleneck and thus should exchange information with each other. The SBD is responsible for reassigning a flow to a different FG whenever the bottleneck changes.

2.5.2 The Flow State Exchange

The FSE can be described as a manager that maintains information exchanged between the flows and calculates a bit rate for each flow based on all the information gathered.

It can be implemented in two ways: *active* or *passive*. In the active version, the FSE will actively initiate communication with each flow and SBD. While in the passive version it does not actively initiate communication and only has the task of internal state maintenance. The FSE keeps a list of all flows that have registered with it. For each flow the FSE will store the following:

- A unique number f to identify the flow.
- The Flow Group Identifier (FGI).
- The priority value P(f).
- The rate used by the flow which is calculted by the FSE in bits per second FSE_R(f).
- The desired rate of the flow, DR(f).

The priority value P is used to calculate the flow's priority portion out of the sum of all priority values. The desired rate might be smaller than the calculated rate, e.g. because the application wants to limit the flow or simply does not have enough data to send. If there is no desired rate value given by the flow it should just be set to the sending rate provided by the flows congestion control.

For each FG the FSE keeps a few static variables:

- The sum S_CR of calculated rates for all flows in th FG.
- The sum S_P of all priorities in the FG.
- The total leftover rate TLO. This is the sum of leftover rate by rates limited by desired rate.
- Aggregate rate AR given to flows that are not limited by desired rate.

Every time a flow's congestion control normally would update the flow's rate they carry out an UPDATE call to FSE instead. Through the UPDATE call they provide their newly calculated rate and optionally a

desired rate. Then FSE calculates rates for all the flows and sends them back. When a flow f starts, FSE_R is initialized with the initial rate calculated by f's congestion controller. After the SBD assigns the flow to an FG, it adds its FSE_R to S_CR. The desired rate is smaller than the calculated rate when the flow is limited by an application, otherwise it will be the same as the calculated rate.

Active FSE In the active version FSE recalculates rates and notifies all the other flows in the FG as well whenever there is an UPDATE call from a single flow.

In [4] there are two examples of active FSE algorithms outlined. The first algorithm The second algorithm which improves upon

Passive FSE In the passive version of FSE it only updates the sending rate for the flow that makes the UPDATE call and not the rest, as opposed to active FSE.

2.6 Shared Bottleneck Detection

The SBD is an entity that is responsible for determining which flows are traversing the same bottleneck. In [4] three methods for deriving if flows share the same bottleneck are mentioned.

2.6.1 Multiplexed flows

One way is through comparing multiplexed flows. Since the flows with the same five- tuple will be routed along the same path, SBD can assume that they share the same bottleneck. However this method cannot be used for coupled congestion controllers with one sender talking to multiple receivers, given that they will not have the same five- tuple. Since WebRTC uses both SRTP and SCTP multiplexed on UDP, this implies they have the same five- tuple and that the first method will work.

2.6.2 Measurement

One might also use measurements of e.g. delay and loss and look at correlations to derive if flows have a shared bottleneck.

2.6.3 Configuration

Chapter 3

Design

Chapter 4

Implementation

Chapter 5

Evaluation

5.1 Testbed

5.2 Experiments

Chapter 6

Conclusion

6.1 Research Findings

6.2 Further work

6.3 Closing remarks

Bibliography

[1] Hari Balakrishnan and Srinivasan Seshan. *The Congestion Manager*. RFC 3124. June 2001. DOI: 10.17487/RFC3124. URL: https://rfc-editor.org/rfc/rfc3124.txt.

[2] Stefan Holmer et al. *A Google Congestion Control Algorithm for Real-Time Communication*. Internet-Draft draft-ietf-rmcat-gcc-02. Work in Progress. Internet Engineering Task Force, July 2016. 19 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02.

[3] S. Islam et al. 'Managing real-time media flows through a flow state exchange'. In: (Apr. 2016), pp. 112–120. ISSN: 2374-9709. DOI: 10.1109/NOMS.2016.7502803.

[4] Safiqul Islam, Michael Welzl and Stein Gjessing. *Coupled Congestion Control for RTP Media*. RFC 8699. Jan. 2020. DOI: 10.17487/RFC8699. URL: https://rfc-editor.org/rfc/rfc8699.txt.

[5] Safiqul Islam et al. 'Coupled Congestion Control for RTP Media'. In: *SIGCOMM Comput. Commun. Rev.* 44.4 (Aug. 2014). ISSN: 0146-4833. DOI: 10.1145/2740070.2630089. URL: https://doi.org/10.1145/2740070.2630089.