MASTER'S THESIS

A Flexible Adaptation Framework for Real-Time Communication

Karl Öhman
2014

Master of Science in Engineering Technology
Computer Science and Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Abstract

Real-time video rate adaptation has become an active topic with the arrival of video communication through the web browser. The most used rate adaptation algorithm in this area is one developed by Google and is being used in Chromium (and Google Chrome). That algorithm, as well as other alternative frameworks and algorithms available today, have a tendency to focus on a specific metric. At high level, the set of requirements for a rate adaptation system includes quick reaction times, fairness to other network traffic and good performance in heterogeneous networks.

This thesis presents a comparison of some current adaptation systems and a list of requirements and metrics for such systems. Also, the design of a new framework for rate adaptation and congestion detection is presented, where the main focus is to use several different metrics together to detect network congestion and compute an appropriate sending rate. A test implementation with focus on delay based adaptation was developed and used in a series of tests to compare it with Google's algorithm and to draw conclusions of performance and design decisions.

The results indicate that the test implementation was able to compete with Google's algorithm in the given test cases. It also revealed the importance of making good adaptation decisions based on the detected issues. Weaknesses of existing strategies are revealed and it is discussed how these can be solved.

Preface

First I would like to thank Ericsson Research in Luleå for the opportunity to do my thesis with them. I would also like to thank my external supervisor Zaheduzzaman Sarker for his support and encouragement which made me work hard and strive further.

I would also like to thank my internal supervisor Olov Schelén who helped a lot with his outside perspective when writing this report.

The topic of bit rate adaptation for real-time media is an interesting one and even though I have spent a lot of time with it, I feel like I have only scratched the surface. It is an old topic stretching back to the birth of the Internet, but has been evolving with the changes in infrastructure and the arrival of new communication services. As I have finished my work within this active research area, others will continue working and develop new solutions and new ideas.

Contents

1 Introduction

This thesis describes work done in the field of congestion detection and rate adaptation of real-time media with a focus on real-time conversational video. Recent developments have seen a focus in peer-to-peer video where the central server is only present during the session setup. After the session is established, the participating clients handle media transmission and rate adaptation themselves. For this to work, all clients are symmetrical and use the same functionality when sending and receiving data. In order to make the subject more manageable, this thesis has limited itself to consider video going from one of the clients to the other. The conclusions drawn can also be applied on multi-directional video with some considerations. The thesis begins with describing the general problem of network congestion before moving on to discussing relevant technologies and other existing solutions and algorithms. It then moves on to describe a new rate adaptation framework design and implementation. The main idea of the framework is to use the combined input from several different metrics to detect network congestion and to react appropriately. The implementation is focused on primarily measuring changes in delay and round-trip time (RTT) together with packet loss measurement. Lastly, tests are described and results analyzed with conclusions and discussion.

1.1 Problem description

Congestion in the Internet is a well-known issue. It can occur when a node in a network is unable to handle packets in the same rate as they arrive. The buffers fill up, which introduces delay and packet losses if the buffers fill up completely. Both delay and loss have a negative impact on the user experience. One specific case is transmissions over radio networks where a number of different factors may affect the signal and cause both delay and packet loss. In many cases it is acceptable that packets arrive late or require retransmission from time to time. Both audio and video suffers when faced with these issues. In the case of real-time video, lost packets can result in a missing or corrupted video frame and delayed packets may contain a video frame that is no longer relevant. Streaming video can partially avoid this problem by using a buffer at the receiver. Audio may be stuttering and have the wrong pitch. An increased delay might cause voice and video to go out of synchronization or delay everything to the point of it not being real-time anymore. Very late packets are as bad as lost, lost packets affect the video quality and this leads to a lower quality of experience for the user. When comparing audio and video, video communication is more susceptible to congestion due to higher required bit rates. This thesis is thus focused on real-time video adaptation.

A real-time video system has to be able to detect congestion and react accordingly in order to maintain a good user experience. The solution is to send less data over the network until the congestion disappears. In the case of real-time video, it can be done by lowering the bit rate by lowering quality of each video frame or dropping certain frames. The difficulty lies in being able to detect congestion and deciding how to react. A system that resorts to sending one frame per second after a temporary increase in packet loss may handle the congestion problem, but the user experience would be terrible.

The desired system should be able to detect congestion before it affects the user experience and should adapt while still preserving a reasonable level of user experience. The system must not be too defensive and give up too much bandwidth but still not be too offensive and demand more bandwidth than necessary and starve other connections.

There are several existing frameworks that address the problem. Among these are systems designed by Google. It is not yet documented how these behave in different kinds of networks and how they behave when competing over the same congested link. Would it be possible to design a new framework that is more adapted to mobile networks and improves over existing ones, with regards to quality of experience, rate adaptation and TCP fairness? TCP fairness is a term used to describe how a certain protocol is interacting with competing TCP streams. A TCP fair protocol would use a similar amount of bandwidth as a TCP stream in the same situation. For example, if a TCP stream detects packet loss, it will lower the sending rate by a certain amount before waiting a while and then attempting increase. A TCP unfriendly protocol operating in the same network might take advantage of this known behavior to obtain additional bandwidth and force out the TCP session.

1.2 Goals

Below are the five main goals for the thesis. The indented lines represent sub goals and deliverables.

- Reach a deeper understanding of rate adaptation by looking at existing solutions and discussions of the topic.
  - Describe and discuss the current state of the art solutions available.
  - List and evaluate the requirements of a rate adaptation framework in a mobile environment.
- Perform tests of the Google framework and see how it handles the requirements defined previously (the Google solution is already present in Chromium). The tests would include unstable connections over different kinds of networks as well as competition with other traffic.
  - Present the results of the tests where the focus is on issues related to mobile networks.
- Design a framework or module with the goal of addressing the issues revealed during the testing.
  - The design process and reasoning behind the decisions will be discussed in the report.
- Implementation of the new framework in the Chromium browser.
- Perform the same tests as before using the new modules.
  - The tests should reveal if the new solution works better than the one designed by Google.
  - Compare and discuss the results.

2 Background

This chapter describes relevant technologies and protocols, earlier research and other systems as well as a more detailed description of the Google algorithm.

2.1 Related technologies

Real-time communication can be performed on a number of different platforms and over several different protocols. A recent development that has grown rapidly is communication between web browsers using HTML 5 [41] and WebRTC [42]. The idea is to move all communication, encoding and adaptation to the browser itself, letting users create their own communication systems using JavaScript. This would remove all need for Flash or other plugins and if all browser manufacturers follow the standard being developed by the RTCWEB IETF group [43], any browser could be used.

Media transport is performed using the Real-time Transport Protocol (RTP) [35] over User Datagram Protocol (UDP) [44]. UDP itself is a best effort protocol, meaning it will send data as fast as it can, but does not guarantee that the data will arrive at the receiver. This responsibility is left to the application. The application requires more information for that to be possible. RTP is used to provide this. It consists of a payload and an RTP header. The header contains information such as send time, sequence number, payload type and SSRC. The SSRC (Synchronization SouRCe) number is used as identification to separate packets from different streams.

QoS related information and various feedback messages from the receiver to the sender are also sent over UDP, but using the Real-time Transport Control Protocol (RTCP) [35]. These messages contain a lot of different information. Amongst other things are an SSRC for identification, number of received packets and number of lost packets. The final transmission component required is a way to transmit a recommended bit rate. This can be done by using Temporary Maximum Media Stream Bit Rate Request (TMMBR) or what is currently used in Chromium; RTCP message for Receiver Estimated Maximum Bitrate (REMB) [36]. The purpose of the two systems is to let the receiver notify the sender of the current estimated available bandwidth (as calculated by the receiver). REMB provides additional support for several parallel media streams between the same clients. This exact case is not considered in this thesis. The feedback from RTCP and REMB messages can be used by the receiver side to inform the sender of the current situation or to control it by sending bit rates that have to be followed.

2.2 Earlier research and other solutions

This section describes a number of existing rate adaptation systems and solutions.

2.2.1 TCP rate adaptation and packet loss detection

The first and simplest example of rate adaptation can be found in the Transmission Control Protocol (TCP). The main priority of TCP is for all packets to arrive at the receiver. A detected loss leads to the packet being resent. As packet losses increase in a congested network, more packets would be resent, further increasing the congestion. RFC 896 from 1984 describes the problem as a congestion collapse, where most packets in the network are retransmissions and no useful traffic is getting through.

The solution, described in RFC 5681, was a combination of algorithms including a slow start and lowering of the sender bit rate if packet loss is detected. Since the main priority of TCP was for all packets to arrive at the receiver, packet loss detection and retransmission at a lower rate solved the problem at the time. These solutions were implemented in TCP itself and anyone developing a system using TCP would not have to worry about packets not arriving properly.

The arrival of best-effort solutions such as User Datagram Protocol (UDP), where the only priority of the protocol was to send the packets to the receiver and hope that they would arrive, moved the congestion control algorithms to the application layer. An example of a rate adaptation algorithm for best-effort flows is TCP Friendly Rate Control (TFRC) which calculates a TCP friendly bit rate based on packet loss and Round-trip time (RTT) [1]. TFRC has a lower variation of throughput over time when compared to TCP which according to [1] makes it suitable for streaming media. It has been shown that the TFRC adaptation is overly pessimistic and result in under-usage of the available bandwidth [2 3]. It has seen some usage in client-server models [4].

Real-time communication introduced new demands on delay and bit rate smoothness. Long or varying delays were no longer acceptable.

2.2.2 Recent developments in real-time video rate adaptation

Research in the topic of real-time media rate adaptation is often concentrated into solving specific issues. This section describes a number of existing or proposed algorithms with different approaches to the problem.

Recent rate adaptation frameworks have shifted focus from detecting packet losses to detecting changes in packet or frame delay. One way of doing this is to send out probing packets with a known spacing at the sender. The receiver can then compare the arrival times and look for signs of increased delay [5] [6] [7]. This strategy appears to have died out in favor of analyzing the content packets instead. Baruza et al present a solution called TREND in a paper where they suggest that packet losses in a stable network only appear following an increase in delay where the network devices cannot handle the incoming packets fast enough [3]. Queues in the network devices will start to build up, meaning that it will take longer time for each packet to arrive at the receiver. Measurement of the changes in delay can be used to increase or decrease the sending rate. The argument put forward for this design is that if delay changes can be reacted to quickly enough, congestion losses could be avoided. TREND monitors the inter-frame delay and lowers the bit rate before packet losses can occur. The authors claim that their solution utilizes 90-95 % of the available bandwidth within 15 seconds [3].

Wireless networks introduce new problems that are not present in wired networks. The most important one is how the radio signal can be affected by various factors such as weather, temporary obstructions or so called hidden terminals, as described by Wong and Starsky [8]. Several frameworks such as RAM [9] monitor the Signal-to-noise ratio (SNR) of the wireless network in order to detect changes in signal strength. That and RAM solve the hidden terminal problem by using Request to send and Clear to send (RTS/CTS) messages. The basic idea is that a node that wants to send a message has to ask the recipient for permission by sending an RTS message, even though no other traffic is detected by the sender. The receiver may respond with a CTS message if no one else is currently sending. At this point, any other nodes detecting these RTS/CTS messages will wait before attempting to send data. Common features for rate adaptation systems for wireless networks are a reliance on information from lower network layers and the assumption that the network consists solely of wireless nodes.

Several recent papers suggest active usage of forward error correction (FEC) to handle packet losses that cannot be avoided by lowering the sender bit rate [10] [11] [12]. Ellis et al compare several different FEC techniques and concludes that most can compensate for losses up to 10 % without significantly increasing delay [12]. Higher losses can be handled but at higher delays.

A somewhat novel solution presented by Singh et al suggest that RTP can be extended into a multipath system that shifts away from congested links without changing the bit rate [13].

In another paper, Singh, Ott and Curcio present the C-NADU algorithm where the sender calculates a suitable bit rate based on an extended RTCP report from the receiver [14]. The algorithm differentiated between congestion avoidance, congestion mitigation and underflow. Tests were made in a 3G environment including this algorithm, a TFRC [1] based algorithm and a number of TMMBR [16] based solutions where the receiver calculates a temporary maximum bit rate and reports it back to the sender. The tests revealed that TMMBR assisted by information of the uplink and downlink provided the best bandwidth utilization. C-NADU proved to be almost as good while the TFRC algorithm provided the least amount of bandwidth utilization. The paper indicates that it is possible to design a rate adaptation system that works in a wireless environment without having to rely on information from the hardware.

A system designed by Alvestrand et al [**Error! Reference source not found.**] suggests a combination of the systems described in [14]. The receiver of a video stream considers the inter-arrival time of video frames and sends Receiver Estimated Maximum Bit rate (REMB) messages back to the sender together with RTCP reports. The sender calculates a bit rate to use based on the information it receives. The network is considered uncongested if the reported packet loss rate is below 2 %. In this case the sender increases the bit rate. If the packet loss exceeds 10 %, the network is considered congested and a lower bit rate is calculated. A loss percentage in between the two limits leaves the bit rate unchanged. The sender bit rate is always limited by the receiver rate (higher limit) and a TFRC rate (lower limit). This is the current rate adaptation framework used in the WebRTC implementation in Google Chrome. Some criticism of this framework is expressed regarding issues with low amounts of feedback messages and problems with self-fairness [18 19]. Feedback messages are sent from the receiver back to the sender to inform it of the current state of the media stream or suggest new sending rates. Self-fairness describes a behavior where two or more media streams share the capacity in a bottleneck equally.

LEDBAT [20] is a rate adaptation system designed for a different purpose than real-time communication. It is intended for peer-to-peer applications that work in the background and seeks to avoid interfering with anything else the user might be sending on the wire. The algorithm requires the sender to attach a timestamp to each packet, which is used at the receiver side to compute changes in queuing delay. This is reported back to the sender, which computes the proper sending rate. The paper argues that a delay-based framework would detect congestion before TCP, which reacts on packet losses. While the goal of LEDBAT is to stay out of the way of TCP traffic it raises an interesting point: Even though a delay-based algorithm reduces its bit rate, TCP flows on the same link would continue to increase their bit rates until packet losses occur.

A parallel but related research topic regards the efficiency of video codecs. [21] and [22] highlights the advantages and suggests ways to improve their efficiency. This research is relevant to the topic of rate adaptation as it can lead to lowering the amount of data that needs to be transmitted.

Congestion detection is not only performed in the endpoints of a network, but also in the network nodes. Active Queue Management (AQM) is a strategy implemented in routers that monitors the buffer space and either drops or marks some packets when the buffers are getting full. This in itself does not solve the congestion problem. The endpoints must still perform detection and adaptation. The marking strategy is known as Explicit Congestion Notification (ECN) [23]. Systems that are able to detect ECN markings could help reduce or remove the need to drop packets at the network nodes. One of the first algorithms in this area is RED [24] and a more recent example is CoDel [25]. A recent algorithm designed by Cisco suggests that ECN markings and packet losses can be used in delay based rate adaptation by assigning certain delays to marked or lost packets [26]. This algorithm also considers PCN [34] markings. The authors present a solution to how to combine different kinds of metrics into one value. They suggest that marked or lost packets are translated into certain delays. Observing packets marked by the network should be considered the cutting edge of congestion detection systems as ECN is yet to be fully implemented in most networks.

Singh, Baruza and Alvestrand discuss similar techniques regarding congestion detection and adaptation. Singh and Baruza show that systems based solely on receiver feedback can work in both wired and wireless networks [2] [3]. The algorithm designed by Alvestrand et al [**Error! Reference source not found.**] is backed by Google and aspires to become a standard, but does not provide statistics for how it behaves in different situations. It appears to be the best suited existing framework for this problem and it will be analyzed further and compared with the implementation in this thesis. It is an example of how two congestion strategies can be used together. This study will also be useful for others as WebRTC support is to be implemented in several web browsers and other systems where effective rate adaptation will be vital.

2.2.3 Reflections on other frameworks

The study of existing solutions indicates that there is a wide array of possible ways of detecting congestion. A flexible framework would utilize several of these detection methods in order to be able to react as quickly as possible. The Google rate adaptation algorithm indicates that a combination of congestion detection strategies is a viable strategy.

Inter-frame or packet delay can often detect possible congestion at an early state, giving the system time to react. A probing packet solution is undesirable as it would result in more packets in the network when a real-time video system is already sending packets with a known spacing. Results from papers describing delay based systems indicate that it is a valid strategy that should be incorporated in a rate adaptation framework with multiple detection methods.

Packet loss detection is important as high losses indicate a problem that requires immediate action. There are also cases such as when switching between access points, where packets may be reordered or lost, but the frame delay remains stable. Relying on packet loss alone will however result in a system that is slow to react and the video quality will be affected before any rate adaptation will take effect.

Being able to detect if the network is actively performing congestion avoidance is a useful addition. An AQM scheme where the network drops packets will be detected by monitoring packet loss, but a marking scheme (e.g., ECN) will require explicit monitoring. (draft-carlberg-tsvwg-ecn-reactions-03) suggest that detected marked packets should be treated as losses in congestion detection schemes. A packet loss detector would thus have to aggregate both lost and marked packets. Such an approach may be viable, but ECN marked packets are also an explicit sign of congestion that could not be caused by anything other than a network device detecting congestion (corrupted IP headers aside). Thus a system that is able to detect these markings should act immediately.

Knowledge about signal strength when on a wireless network could be useful as lower signal strength may indicate a coming access point switch or an increase in loss rate. This information cannot be counted upon since the client might not be in a wireless network all the time. One should also note that the sender is not aware of the signal strength of the receiver and vice versa.

FEC usage could be adjusted dynamically based on detected network conditions. If it is possible to separate losses caused by congestion and losses caused by, for example, a wireless link with poor reception one could increase FEC when losses are not caused by congestion. Finding the best codecs and FEC strategy for the situation is out of scope of this thesis, but it affects the overall performance by changing the amount of acceptable losses and the amount of data needed for each frame.

Knowledge about and access to some of these parameters will vary depending on platform and network. For an application layer system that can be run over any network, it is possible that metrics from the incoming video packets are the only available parameters.

2.3 Overview of the Google congestion control algorithm

The Google congestion control algorithm consists of a sender side module and a receiver side module. The sender side uses a packet loss based congestion detection system while the receiver uses a delay based one. The sender uses the receiver side bit rate estimation as an upper bound. The algorithm is described in [17] but the specification is rather high-level and some details differ in the implementation.

The choice of using the Google algorithm as a comparative implementation is based on several factors. It is currently implemented in the Chromium and Google Chrome browsers and is thus actively used. The Chromium platform allowed for running tests of Google's solution and the one described in this thesis using the same platform. It also prevented any mistakes that may have been made when implementing it based on a design document.

2.3.1 Sender side

The purpose of the sender side rate control module of the Google algorithm is to calculate a target bit rate for the video encoder to follow. This rate is based on feedback from the receiver. The feedback consists of a REMB rate, which specifies the estimated maximum bit rate that the receiver can handle, and RTCP reports with packet losses and round-trip time. The responsibility of the sender side system is to monitor packet losses and adjust the sending rate accordingly while also taking the REMB rate into account.

The sender stores incoming REMB rates and will interpret the network conditions as changed if the incoming rate is lower than the current rate. If higher, the REMB rate will be stored until the next packet loss report arrives and a new sending rate is to be computed.

Upon receiving RTCP reports with RTT and packet loss ratios it computes a new sending rate based on the amount of loss.

After a new rate has been computed, the resulting rate is compared with the last received REMB rate. No matter what rate the sender has calculated, it must not exceed the receiver's maximum rate. If the rate has been lowered because of high packet loss, it will be limited by a lower bound set by a TFRC rate. It will not have this lower limit if the rate is lowered because of a low incoming REMB rate. TFRC stands for TCP Friendly Rate Control and is described in RFC 5348. It specifies a rate calculation for best-effort flows that is reasonably fair when competing with TCP flows over a congested link. The motivation for the loss thresholds put forward in the IETF draft is that if some losses are caused by overuse, they will eventually increase above 10 %. If the losses are unrelated to congestion they should not increase and no action needs to be taken.

Constant values such as minimum bit rate, maximum bit rate and the starting rate are specified both by the video codec and the video engine. The video engine specifications for the media are 50 kbps, 2000 kbps and 300 kbps respectively.

2.3.2 Receiver side

The purpose of the receiver side is to monitor the incoming video stream and look for changes in frame delay. The receiver uses the information to calculate a new receiver maximum rate estimation and notifies the sender via a REMB message.

When the video receiver module receives an RTP packet, it is sent to a component called the RemoteBitRateEstimator. This in turn, passes the information on to the OveruseDetector which checks if it has a complete frame. If that is the case, it updates its state using a Kalman filter. This state is kept until a new frame is completed and a new state is calculated. If it does not have a complete frame, the information is stored and the OveruseDetector waits for the next packet to arrive.

If the state calculation results in overuse, a new bit rate estimate calculation is triggered. New bit rates are also calculated periodically, which may result in an increase of the recommended rate.

The system relies on RTCP with extensions for TMMBR and REMB for receiver feedback. REMB was recently chosen over TMMBR for receiver based bit rate estimation feedback. The reason for this choice was to improve functionality for multiple video streams from the same browser client. This specific case is not considered in this thesis.

3 Framework design

This chapter describes the design of the suggested framework. It begins by describing the basic principles and requirements before moving on to components and system behavior.

3.1 Basic principles

This adaptation framework is intended to be used for congestion detection and rate adaptation of real-time media streams over a heterogeneous network. The purpose of this framework is to provide a target bit rate that can be used by a media stream application on a certain network without causing congestion. The rate is based on detected feedback from the media streams as well as from the network itself in the form of ECN markings.

Real-time video communication is performed using RTP packets for media streams and RTCP packets for receiver feedback with REMB extensions for transmitting bit rates. The framework expects that it is possible to observe RTP streams and that feedback over RTCP is available for use. It does not specify any specific codecs as long as the one used are able to adapt their encoding rates.

The framework also considers input from the user or application regarding minimum and maximum bit rates, as some applications may not require the highest available rates, while others only operate on high-speed LANs where congestion is less of a problem. Another form of input that is considered is letting the sender prioritize between several media streams in cases where there is not enough bandwidth for all of them. This input is considered at the sender side control module. It could also be decided at the receiver side if an implementation requires the receiver to be in complete control over the session.

The adaptation framework consists of several components that can be combined and implemented in different ways depending on application requirements and environment where the application will be used. The relation between the application, network and the framework can be seen in Figure 1.

3.2 Requirements

The following requirements should be followed by a good rate adaptation system. The priorities are represented by the keywords Must and Should. This list turned out to be similar to the requirements stated by the RMCAT IETF working group. This indicates that the understanding gained by doing research during this study is shared by others.

1 Must prioritize low delay over high quality in the event of congestion.
   - There is always a tradeoff between delay and quality but a real-time video feed of lower quality is preferable over a delayed one with higher quality. Refusing to reduce the video quality when facing congestion will inevitably lead to increasing delay for everyone sharing the bottleneck, not just the video application. This requirement does not imply that the bit rate should go down to minimum without attempting to compete for bandwidth. An example would be to slightly lower the bit rate and assuming that others will do the same.

2 Must react based on information from several sources, including delay, packet loss and lower level feedback such as ECN.
   - Network issues can manifest in several ways and different systems are better at detecting some issues than others. A flexible solution would be able to take several information sources into account to detect congestion. Previous systems have a tendency to focus on one source only. Previous solutions have been focusing on single sources of feedback and point out how it can solve a specific problem.

3 Must be quick to react to network changes and persistent congestion
   - The system must be able to quickly detect and handle congestion and not overreact to small changes. Being able to quickly detect congestion is vital for an effective rate adaptation system.

4 Should be able to differentiate between packet losses caused by congestion and other losses that may be unrelated to congestion.
   - Attempting to solve a packet loss problem that is caused by an unstable link or faulty hardware by lowering the sender bit rate would be futile as there are no queues to shorten. Some sort of redundancy or error correction could be used in cases with low delay and high losses. A low delay indicates that there should be room for extra packets to be sent.

5 Should keep a record of the rate history of specific connections
   - Should keep track of the rate history in order to improve startup efficiency and avoid unnecessary attempts to increase the bit rate beyond a maximum for a certain link.

6 Must work together with different kinds of other traffic (web surfing, YouTube).
   - Other user activities performed in parallel with real-time video chatting can result in a number of different kinds of traffic flows. Web surfing usually generates several short bursts of TCP packets while streaming services such as YouTube produces longer streams of packets. The system must not decrease its rate too heavily, but rather make sure to adjust to a fair rate. A requirement for this is of course that the other flows utilize rate adaptation as well.

7 Must work in different kinds of networks (wired networks, 3G, LTE and Wi-Fi).
   - Users of today move around a lot. If the system using the rate adaptation module can handle switching between networks or switching from Wi-Fi to a 3G network, so must the rate adaptation. Such a switch may cause temporary packet reordering or losses, but may also significantly change the network conditions.

8 Should take user preference on quality/delay into account.
   - If the user is satisfied with a lower media quality, settle on that instead of trying to use more bandwidth than necessary. This could also be set automatically by the client side device based on screen size. Input from the user could be used as input to the quality vs. delay issue.

9 Should not generate more feedback messages than necessary.
   - Large amounts of feedback messages may become a source of congestion in itself when the throughput is limited. Too few feedback messages will on the other hand lead to a system that is slow to react to changes in the network. There are several theories on how this should be done. The problem is discussed from the TCP point of view in [31].

10 Must listen to feedback messages and be able to cope with missing ones, indicating congestion in the feedback channel.
   - The receiver must send feedback messages to the sender with information about the detected network conditions. This information could be a new sender rate, feedback that the sender needs to calculate a new rate or a combination of both. Missing feedback messages should be interpreted as a sign of congestion. If no messages arrive it should be safe to assume that the link is heavily congested. Even if there is nothing wrong with the network it would be irresponsible to send data at a high rate without receiving feedback as an actual congestion event would go unnoticed. A single lost feedback message should not cause a decrease in bit rate.

11 Should include verification of the sender of the feedback messages.
   - There is a possibility for a third party to send false feedback indicating congestion when there is none. Guarding against a third party that can drop or modify packets is unnecessary as nothing could be done to help it. An attack that consists of flooding the network with packets would be handled like normal congestion. But as usage of real-time video communication increases, a wide-spread program that listens to ongoing traffic and proceeds to sabotage it could pose a serious threat. This requirement is not a Must, but it is something that should be considered.

12 Should support input from other rate adaptation algorithms.
   - If one system is detecting changes in the network that another one fails to detect, the first one should be able to notify the other system. This would still have to take the above point into account. This is also not a Must, but sharing information between several systems in order to maintain a good experience for all participants is relevant to everyone. This point would require that any cooperating systems are using the same protocols.

The design process began by specifying a set of requirements for rate adaptation systems for real time video. These requirements were specified by me based on what I have learned from studying earlier solutions and following the discussions of IETF working groups. Standards and requirements within this area are vital as congestion avoidance in a network only works if all participating flows are able to detect congestion and adjust their sending rates. Among the more important requirements are fairness to other flows and the ability to cope with missing feedback messages. An exact definition of fairness is yet to be decided, but an example of it is that no single flow should decrease its sending rate at a slower pace than others and by doing so obtain an unfair advantage.

3.3 Components

The proposed framework consists of several components responsible for detecting different network issues. The components are meant to be independent of each other in order to let the implementer choose the parts that are relevant for the specific system. This allows the user or other parts of the client system to directly affect the behavior of the rate adaptation algorithm. This feature is not present in most other rate adaptation systems. The point of a flexible framework is to provide a number of different metrics based on situation and implementer preference.

In theory, single metrics could be used, but the performance would suffer unless the system is used in specific networks where the possible conditions are known beforehand. Less efficient methods could include only monitoring packet loss or even let the user lower the bit rate manually if he feels that the video is getting choppy (not a good idea).

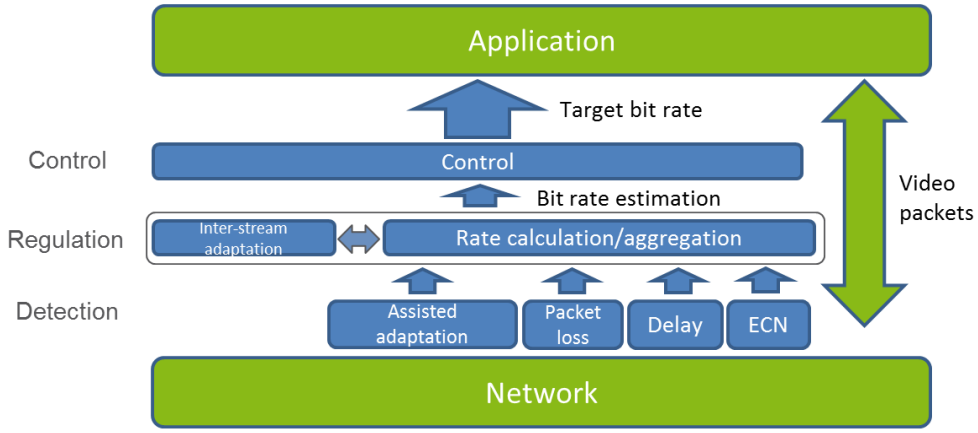

Figure 1 Figure showing the different framework components (blue) and how they relate to the application (green).

Aside from the components described in this section, any number of different congestion detection and avoidance solutions may be added due to the modular design. The simplest form of extension is to add components that can detect congestion and produce a suggested bit rate based on the current rate and the current network conditions. Other solutions might add additional information to existing modules in order to increase their efficiency.

3.3.1 Sender side control module

This is the main module of the sender-side. The sender side will be driven by received RTCP reports and may calculate or adjust the rate and notify the video encoder every time new feedback arrives. This module is specific to the sender side as it is responsible for communication with the video encoder. It will enter a Panic mode if RTCP packets stop arriving and will lower the bit rate until the RTCP reports resume.

This module should be aware of multiple streams sharing the same link and allow the user or application to prioritize between them.

This module is expected to output a bit rate that can be used by the video encoder.

3.3.1.1 Listen mode

This is the default and preferred mode of the sender. It receives feedback from the receiver and reacts to it. The sender remains in this mode as long as feedback messages arrive as they should. When using RTCP, feedback messages may be sent at varying intervals (as suggested in RFC 4585). The feedback messages are expected to have a minimum frequency at which they are sent. This minimum frequency can together with the measured round-trip time be used to detect when receiver feedback is missing. RFC 3550 recommends a maximum interval of five seconds.

A system using per-packet feedback would have to use other thresholds.

3.3.1.2 Panic mode

If RTCP packets stop arriving, the network is assumed to be heavily congested and the sender enters Panic mode. The rate is decreased rapidly and held at a minimum level until RTCP packets start arriving again. This mode exists to ensure that rates do not go uncontrolled, but still lets the session continue. At this point it is up to the participants of the video session if they wish to wait for conditions to improve or terminate the session. Possible reactions are to either pause the video stream or to terminate the session. RFC 3550 specifies that participants of a session using RTP/RTCP must check if other participants have timed out. Timeout is defined as if a participant is not heard of within a specified number of RTCP intervals. A further study into when sessions should be aborted was performed by the AVTCORE IETF working group and presented in [50].

Increasing the rate without feedback is not an option.

3.3.2 Packet loss components

The packet loss based component is split into two parts, the detection and the rate calculation.

When used together with other congestion indicators, the loss rate would ideally not be able to go high enough for a big rate decrease as the video quality would have been severely affected at that point. If such high losses do occur, a large rate decrease will be motivated.

3.3.2.1 Packet loss detection

The packet loss detection could be performed by monitoring the RTP stream for missing packets at the receiver side or by monitoring information provided by the RTP header extension at the sender side. Some systems use retransmissions of RTP packets that fail to arrive. Depending on exact implementation, such a packet may be counted as lost. If the loss detector compares sequence numbers and number of packets arrived between times t1 and t2, a missing packet that is resent after t2 will be counted as lost. A system that resends all lost packets could monitor the number of retransmissions instead of losses.

The information gathered should be sent to a rate calculation module directly if located at the same side as the detection component. In the case where the detection is made at the receiver side and the rate calculation is made at the sender side, the loss rate should be sent over RTCP at the regular interval.

3.3.2.2 Packet loss rate calculation

The loss rate is compared to a set of thresholds indicating if the losses are getting too high and if the rate should be decreased.

A decrease in rate could be based on the loss amount with a calculation such as TFRC [1]. Another more aggressive approach is to lower the bit rate by a fixed amount (such as half of it) every time losses occur. Further examples on packet loss reactions are Additive increase/multiplicative decrease and Additive increase/additive decrease.

The packet loss rate calculation should result in a bit rate that either indicates that no reaction is necessary or a rate that is lower than the current sending rate if losses are too high. This rate may either be used locally if the calculations are made on the sender side, or sent via a feedback message (such as TMMBR or REMB) if on the receiver side. If more than one strategy is used to calculate bit rates at the receiver side, it may be sent to a rate aggregation module.

3.3.3 Delay components

Delay detection considers the time it takes for a frame or packet to be sent from the sender to the receiver. Packet delay considers individual packets while Frame delay considers the time it takes for a complete video frame to be sent from sender to receiver. By observing the differences in the delay, one can detect when the queues in the network are growing and adjust the sending rate before the queues fill up enough for packets to be dropped. An example of an implementation of a delay based solution can be found in [2]. Observing frame delay instead of packet delay should provide smoother output as individual packets may be late or lost. Further research into which of packet or frame based delay measurement is the optimal choice could be done, but is not the focus of this thesis.

3.3.3.1 Delay measurement

These components measure changes in frame delay in order to detect when queues in the network are growing larger. The calculations take frame or packet size and arrival times into account in order to get an accurate reading of the changes in delay in the network. The measurements will be filtered to avoid fluctuation as the frame rate may be high and the sizes may vary a lot. This will provide output indicating if the delay is stable, increasing or decreasing. Once the delay increases above a certain threshold and stays there for enough frames, overuse of the available bandwidth is declared.

3.3.3.2 Delay rate calculation

The rate adaptation will operate by considering the current delay situation and the current target bit rate and outputting a new suggested rate.

The measurement and rate calculation can be split into two separate components if necessary, but the coupling is rather strong and they benefit from being located at the same side of the session.

The delay measurement would be done by observing an incoming RTP stream at the receiver side or by observing feedback messages at the sender side. As with the packet loss module, the expected output is a bit rate indicating if the current target rate should be changed or kept stable. If on the receiver side, the resulting rate could either be sent with a feedback message to the sender or be part of more rate calculation before being sent.

3.3.4 Explicit Congestion Notification components

Explicit Congestion Notification (ECN) [29] is a way to let network nodes signal to endpoint nodes that congestion is detected without having to impair the packet flow. Unlike when dropping packets, a marked packet is an unambiguous sign of congestion and can be reacted to immediately. The endpoints must agree on using ECN when a media session is set up. Setup and usage of ECN over RTP is detailed in [33]. A rate adaptation design using ECN together with packet loss and delay is described in [30]. Another example can be seen in [46].

3.3.4.1 ECN detector

This module detects packets with ECN markings in an attempt to detect when queues in the network are starting to build up and the nodes start marking packets. AQM implementations that drop packets are not considered by this module. If the system can successfully detect ECN marked packets it will have a way of knowing when the network is detecting congestion.

3.3.4.2 ECN rate calculation

A detected marked packet should cause the system to react immediately. An ECN marking will only appear (assuming no IP header errors) if a network device has explicitly marked it due to detected congestion. Adaptation techniques similar to those of packet loss based strategies could be used where every marked packet is treated as a loss event. Additive increase/multiplicative decrease and Additive increase/additive decrease approaches could be used here.

Any detected marked packets could also be added to the packet loss measurement as [32] argues that marked packets should be counted as lost packets in congestion control systems. This may not be necessary if this module reduces the bit rate enough when marked packets appear.

The same module could be responsible for monitoring PCN markings. PCN is not yet adopted, but it is proposed that it will be represented by the same bits as ECN in the IP header. [34]

ECN markings could be sent over RTCP in a sender-driven system as described in [33]. If detection and rate calculation is performed at the receiver side, the resulting rate could be sent as a TMMBR message [46] or be sent to an aggregation module if more than one rate is being calculated.

3.3.5 Inter-stream adaptation module

In addition to rate adaptation based on feedback from its own video session, the system would use this component to accept feedback from other sessions if allowed to.

Several media streams that share the same bottlenecks could share rate adaptation system as well according to [27]. Such a module would consider the current active media streams and the available bandwidth for a link and prioritizes between the streams. A system using this feature must be able to take control over other streams, as well as accepting that others take control of it. Being able to assign priorities to the streams would allow the system to either give all streams a fair share or to give some more bandwidth than others.

This module is located in the Regulation layer in Figure 1. In Figure 3, it is part of the sender side control module.

3.3.6 Statistics/history module

This component would keep track of the performance during video sessions. By keeping a record of what bit rates that are achievable during sessions between certain clients, parameters such as startup and maximum bit rates could be set beforehand to improve the user experience. This would let the system avoid having to use a slow start approach and start at a bit rate that is known to be achievable most of the time. This is part of the Rate calculation/aggregation box in Figure 1.

3.3.7 Rate calculation/aggregation module

This component uses the output rates from the different components and determines the appropriate target bit rate. This module could accept input from any kind of rate adaptation implementation that is connected to it. It may also be responsible for increasing the bit rate if no congestion is detected.

A module such as this could be placed on either the sending or the receiver side depending on preference. However, if a module layout where several bit rates are calculated at the receiver side is used, it may be necessary to use a module that can aggregate the rates into a single one that can be sent as a TMMBR message. In that case the sender can be sure that the TMMBR rate it receives is the current maximum rate that a certain receiver is prepared to handle. The sender would still receive several different TMMBR rates in a multi-stream scenario, but these reports would be separated based on SSRC. The alternative would be to let any number of modules independently send TMMBR messages to the sender side, which could work but may be more difficult to handle as high and low TMMBR rates may interleave each other.

If placed on the sender side, this module can be merged with the Sender side control module as can be seen in Figure 3.

3.4 States

The system should operate within a set of states that define how it should behave. The behavior would be the same no matter if it is located on the receiver side or on the sender side. The main purpose of these states is to clarify why and how the system is currently behaving as it is. It lets the system react differently to the same input depending on what state it is currently in. Figure 2 shows how the state transitions in such a system could work.

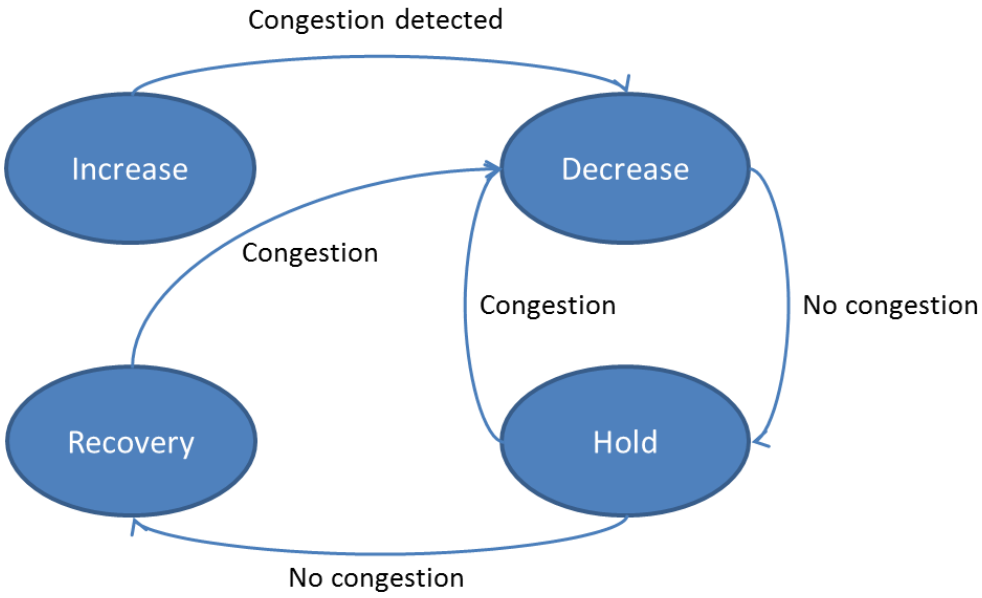

Figure 2 State automata showing how transitions between the system states could work.

The device could automatically detect what network it is in and adjust the behavior according to predefined profiles. Other systems use constant values, but changing demands from the user and application should be met with adaptive behavior. This input would still be limited by what the system is capable of. These features may be included as additional functionality in the appropriate modules. The intention is to be able to make changes to the initial configuration if conditions change.

3.4.1.1 Increase

The system is set in this state when a new session starts. As long as no issues with the network are detected or the system reaches its maximum rate, the bit rate will be increased step by step. If the maximum rate is reached, the system will enter Hold mode. There are two distinct types of increase scenarios; startup and recovery.

3.4.1.1.1 Startup

A relatively small increase factor would let the system detect congestion before it grows too large. The reason for choosing a slow-start approach is to avoid causing congestion by sending too much data before knowing anything about the current network situation. This may be the optimal operation in that case. A shorter call might only last for a couple of minutes and it seems wasteful to spend a large part of the time with low bit rate if the network can handle higher rates. I suggest a set of options that may provide better performance during startup.

- Set a higher starting rate while using a relatively small increase factor. This will not only provide a higher initial bit rate, but will also avoid having to increase the rate at very small steps at the start.
- Use a higher increase factor when the bit rate is below a certain threshold. This threshold could initially be half the maximum bit rate and later during the session a fraction of a recent maximum send rate.
- Maintain a database where known IP addresses can be associated with certain available bandwidth levels based on previous experience or guaranteed QoE.

3.4.1.1.2 Recovery

Recovering from a congestion event is similar to the startup process with the exception that the congestion may have lessened and not disappeared completely. Therefore an increase in this case must be done more carefully. Testing will show how aggressive the recovery rate can be. The intention is to avoid the case where the system starts to increase faster than the network can recover from the previous congestion event. In such a case we could get a saw tooth-like curve instead of the desired smooth increase. On the other hand, a system that waits too long before deciding to recover will waste available bandwidth.

3.4.1.2 Decrease

This state will be entered when congestion is detected and stayed as long as the detected issues (such as delay, loss, ECN markings) are not improving. While in this state, the rate will be decreased.

The new bit rate will be the lower of the following rates

- Rate based on delay measurements

- Rate based on loss measurements
- Rate based on ECN measurements
- Other rates

The system should have a set minimum bit rate. If that rate is not sustainable it should be up to the application designer to decide if the system should keep sending or abort the session.

3.4.1.3 Hold

After the rate has been lowered and the conditions are improving, the system may enter Hold mode where the rate remains unchanged for some time before attempting to increase the bit rate again. The idea is to give the network a chance to empty its buffers as well as avoiding to attempt to increase the rate too often.

3.5 Component placement

An application using this framework may use any of the components that are deemed useful in the given case. This may vary depending on what information is available and what kind of network the system is operating in. For example, in a certain network with huge buffers, observing packet losses may be unnecessary as there should not be any congestion induced losses. An application that intends to be used in a wide variety of homogeneous networks would benefit from including as many detection components as possible. The calculations could also be split into several separate modules or all be performed in a single place. Following is a set of possible placement strategies, along with pros and cons.

3.5.1 Sender side control

In this setup, the decision to change the bit rate is taken at the sender side. This strategy still requires some detection on the receiver side. All information gathered there would have to be sent back to the sender over RTCP. To be able to react to congestion in time it may be necessary to use the AVPF early feedback profile of RTCP, described in [47].The sender is in control and can decide what to do with the available bandwidth. This strategy has an advantage in situations where the sender wants to make sure that a certain media quality is achieved or when it is using several streams and want to prioritize between them. Figure 3 shows a possible design for a sender-based approach. An example of a system using a sender side approach is regular TCP rate adaptation.

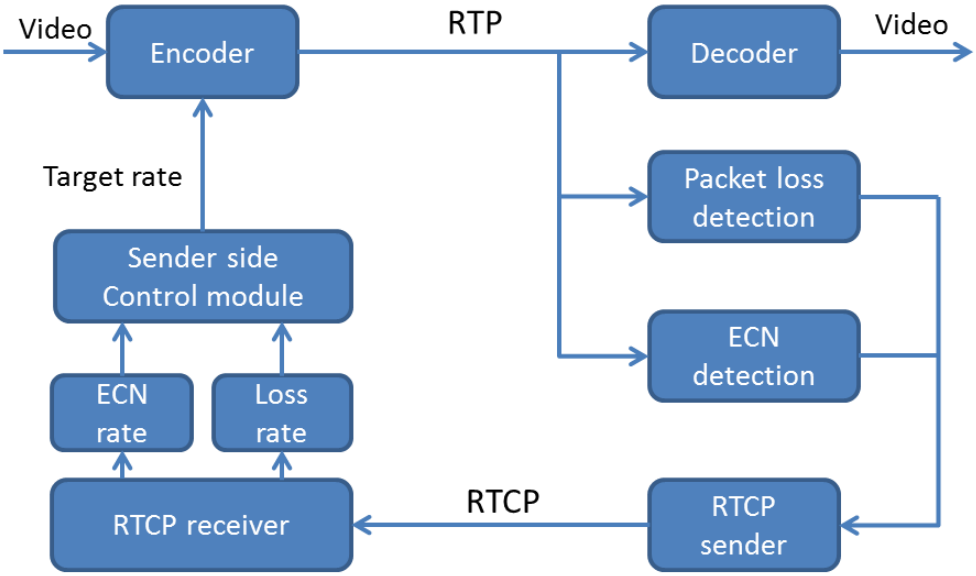

Figure 3 A system using packet loss and ECN as metrics, with detection at receiver side and calculations at sender side. The control module receives two bit rates and decides on a target rate based on those.

3.5.2 Receiver side control

This strategy puts the control at the receiver side. The receiver would calculate an appropriate bit rate based on the gathered information and transmit it to the sender.The sender then adapts to this bit rate. Arguments for this strategy include a reduction in feedback traffic, as it would only consist of a target rate and no other statistics. Another advantage is a reduction in system complexity and the ability to use all available feedback without having to aggregate it into information that can be sent over RTCP. This solution is favored when the receiver should make the decisions regarding acceptable quality levels. In many cases this may be true.

3.5.3 Shared control

This third option is a compromise between the two above. Such as system could for example let the receiver calculate a maximum rate that it thinks is acceptable while still sending available feedback to the sender and letting it decide the actual bit rate. This design would let the sender have control over the session and let him change priorities between several streams or decide if link conditions are acceptable or if the session should be terminated. The receiver still maintains control in the way that it can specify the bounds of the bit rate. A downside with this design is increased complexity where the system maintains separate states at each side.

3.5.4 Design suggestion for an new system

The choice of layout using the framework in this thesis would be a receiver side approach using Packet losses, delay and ECN markings (where available) as described in sections 3.2-3.4. The layout can be seen in Figure 4 .

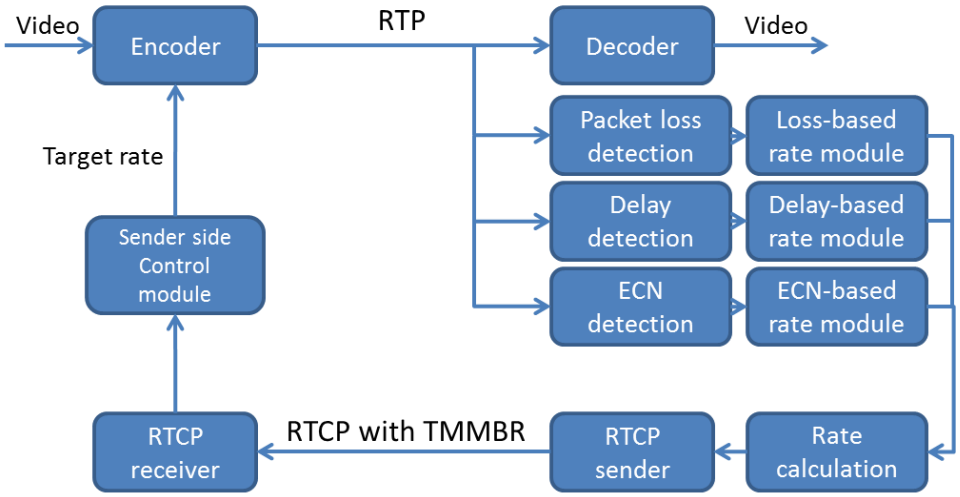

Figure 4 Suggested design using delay, packet loss and ECN as metrics. The rate calculation module considers incoming bit rates from the three other modules and decides on an appropriate target rate that is reported to the sender. The sender considers this rate and may use it straight away or use a part of it if the stream is of low priority.

The receiver would consider these three inputs and let the modules compute one bit rate each. The three rate modules would consider the incoming congestion indications and the current target bit rate and calculate new rates. These rates are compared in the Rate calculation module. If any of the rates are lower than the current target rate, it is chosen as the new rate and is sent to the sender side. If none of the modules indicate congestion, the rate may be increased. The operation is in line with the component description in section 3.7.

The sender side control module considers the incoming rate of this stream as well as others in order to let the client prioritize if the bandwidth is limited. As described in section 3.1, it is aware of the frequency of incoming RTCP reports and will lower the sending rate if reports stop appearing.

The choice of metrics to consider contain the commonly used delay and loss based strategies as well as the not yet common, but upcoming ECN markings. This design would work well in networks without ECN and would work even better where ECN is available. The component placement is motivated by simplicity of design, as well as keeping the amount of congestion control related traffic as low as possible.

3.6 Input and update intervals

Packet statistics are collected as RTP packets arrive. The delay, packet loss and ECN modules will monitor incoming packets and provide relevant information upon request. Updates of the frame delay are made upon every received frame. User and application input that affects system variables, is accepted at any time.

The receiver side rate adaptation is run whenever new input is available. The maximum interval will be one second. This value is shared between the sender and the receiver and will be used at the sender to detect if RTCP packets are missing. Feedback may be sent immediately if a new bit rate is lower than the previous one. This will ensure a rapid decrease in bit rate if congestion occurs. Before sending, the bit rate will be compared with the current minimum and maximum bit rates to make sure it stays within the specified bounds.

The sender side rate adaptation is run every time an RTCP packet is received.

4 Framework implementation

This chapter describes the implementation process of the framework. It touches on implementation details and platform choice.

4.1 Choice of platform

The implementation is based on Chromium version 28.0.1474.0. It was retrieved from the official Chromium repository on 2013-04-11 at 10:30 CET. Visual Studio 2010 was used for compiling. The testing of the Google algorithm was performed using this build as well. The Chromium browser was chosen as a platform for implementing the framework for several reasons. It is open source and freely available to anyone. It already contained the Google algorithm, making it easier for me to evaluate it and understand how the presented solution could be used in its place. Testing the algorithms and being able to see the result in video and not just as numbers and graphs was also helpful. Implementing the framework in a real system that is actively used today also further enhance it as a proof of concept.

Choosing Chromium had downsides as well. The major problem was the sheer size of the codebase and the time it took to compile it. It also made changing internal variables during testing difficult. Another issue was that the implementation would only be able to use metrics available in the application layer. On the other hand, this highlights potential difficulties for others who intend to implement this framework.

4.1.1 Signaling and media transport

Media is transported using RTP [35] packets over UDP. Feedback in the other direction is sent using RTCP [35] packets with REMB [36] extension for bit rate feedback. Google recently switched to REMB from TMMBR.

The time between each sent RTCP feedback packet is based on the current sending bit rate in the same direction according to the formula:

Time (seconds) = x/2 + x * rand(0,1), where x = (360 / [bit rate in kbps])

There is a lower limit of 1 second according to the guidelines of RFC 3550. REMB extensions are added when new output from the receiver side adaptation is available.

As stated earlier, using Chromium meant that the media transport and feedback functionalities were already in place, letting me focus on the adaptation itself.

4.2 Purpose and scope

The main purpose of the implementation was to be able to see what components are viable to implement and if delay based detection can be made in a simpler way than how Google does it. The point is to create a proof of concept to test and illustrate solutions and concepts. To test the viability of the requirements and perhaps providing the basis of a new congestion avoidance system that can compete with the ones that are available today. Peer-to-peer video in the web browser is rapidly becoming more popular. The Google algorithm is at the time of this thesis the main one that is used in commercially available browsers. It is of interest to everyone that more congestion strategies are put to the test in order to avoid just settling for the first one that does the job. The approach was to start small and expand to more modules if possible. Delay-based adaptation was given priority as it is a common solution among other systems today. It is also the main congestion detection method used by Google, whose solution is being discussed a lot recently. Packet loss based detection was also implemented, but control based on this was given lower priority due to time constraints. The lower priority can be motivated as most network devices are expected to have large buffers and a good delay based algorithm should be able to detect congestion and react before the buffers are full. For simplicity and ease of illustrating the results, the testing is focused on measuring and applying impairments to one direction of the video sessions. That is, Client A and Client B are taking part in a two-way conversation. Network impairments are applied on the video stream traveling from A to B. This also allowed for direct visual comparison of subjective video quality during the testing.

4.3 Implemented components

The design allows for great flexibility when it comes to component choice and placement. For this implementation, I chose a receiver-based approach where congestion detection and rate adaptation is performed by the receiver. The implementation consisted of modules measuring delay and packet loss, which were used in a control module to compute a suitable sending rate. The sender side adaptation consisted of a module that received the incoming rate recommendations, and forwarded them to the video encoder. The sender also kept track of the frequency of feedback messages in order to detect if the receiver stopped responding.

4.3.1 Packet loss measurement

Detection of lost RTP packets was done by comparing the number of incoming packets over a certain time with the difference in RTP header sequence numbers. This simple strategy is accurate as long as the packets are not heavily reordered.

Some considerations have to be made regarding rate calculation based on packet losses. A sufficient number of packets must be gathered before a loss percentage can be calculated. If a calculation is made after receiving 8 packets, two missing packets during that time would result in a 20 % loss rate, which is considered very high. If the same two packets were lost when collecting 100 packets before calculating a loss rate, the percentage would be much smaller. On the other hand, collecting too many packets could lead to longer reaction times.

Another consideration is that packet losses are not only caused by heavy congestion, but may appear randomly. One can therefore not assume that any loss is a sign of congestion and should use thresholds to decide if losses are high enough to warrant a rate reduction.

The output rate of this module depends on the level of packet loss. Loss below 5 % does not provoke a rate decrease. Loss between 5 % and 10 % causes the rate to freeze while losses above 10 % cause a rate decrease depending on the percentage.

The reasoning behind the thresholds is that sporadic packet losses should not cause the system to reduce the sending rate, while losses due to congestion are much larger.

During the implementation phase, this module received less focus than the delay based adaptation and for that reason, the test cases were adjusted to reflect that. The loss measurement is performed after the packets have passed the jitter buffer, but some issues due to reordering were still occurring.

4.3.2 Delay measurement

Measuring delay accurately when communicating between two different devices is more difficult. A naïve approach is to compare the timestamp of a received video frame with the current time at the receiver side and observe the difference. This cannot be done as the clocks on two different devices may not be perfectly synchronized.

The solution is instead to compare the differences between send times and arrival times separately of several consecutive frames to observe how the delay changes. This information is used to obtain the inter-arrival time.

Each incoming RTP packet contains a timestamp from the original video frame. This timestamp is used to determine when an entire frame has arrived if it was split up into several packets.

These measurements need to be filtered as they may fluctuate a lot. This is done using a low pass filter. It is of a simpler design compared to the one developed by Google. Part of the testing was to see if they were comparable.

The mean variance of the inter-arrival time is then observed and compared with a moving threshold. If all is well, the mean variance will stay below the threshold and changes will be relatively smooth. The sudden appearance of congestion is reflected in a sharp increase in mean variance that exceeds the threshold. At this point, overuse is declared. This happens when the video frames arrive with increasing time spacing. This occurs when buffers in the network are filling up and implies that the sending rate should be decreased. Underuse indicates that packets are arriving with shorter spacing than they are sent at, reflected in a decreasing mean variance. This indicates that the buffers in the network are emptying and that more bandwidth will be available. When the mean variance is stable and within acceptable levels, the state is said to be normal and the sending rate may be increased if not at maximum already.

Detected overuse causes the output bit rate to drop. The initial drop is set to 90 % of the estimated incoming rate. If this initial drop is not enough, the rate will be lowered step by step based on the previous output rate. The reason for the size of the initial drop is that the incoming bit rate will have been lowered due to congestion and the extra 10 % is used to leave some capacity for the packets that are in the network buffers. Once the sending rate has decreased, underuse will be detected as the queues empty. This indicates that the decrease was successful and puts the system in a hold state. After holding for some time and remaining in the normal state, the system is allowed to attempt to increase the rate.

4.3.3 Receiver side control

The controller responsible for changing the bit rate depending on output from the delay module works in four modes; Increase, Decrease, Hold and Careful.

During Increase, each increase attempt is compared to the incoming bit rate in order to detect if the attempt lead to a higher rate. If that was the case the system would proceed to increase the rate again. If it was unsuccessful the system would return to the previous rate. This behavior prevents increases far above the incoming bit rate and helps reduce the delay. The incoming rate is also monitored and if it has not changed for a couple of increase attempts while the system is in increase mode, the system assumes that it has reached the current maximum available bit rate and holds for a while before trying to increase the bit rate once more. The final version of the system was increasing at 5 percent per step as too large increase could put the system way above the available bandwidth. This risk can be observed in the plots from test cases with a bandwidth limitation in the results chapter. The small increase steps lets the systems react before the sending rate is allowed to go far above the available bandwidth. Larger increase steps would let the sending rate go higher above the available capacity.

The Careful state works similarly to the Increase state, but is more careful. When the Careful state is entered, the current bit rate, assumed to be safe as the system is no longer holding or decreasing, is set as a temporary maximum. Increasing works as mentioned above, but if an increase attempt fails, the system reverts to the temporary maximum. This maximum is increased if the current sending rate is increased high enough above the current temporary maximum.

Two competing theories came up during development. The first idea was to enter the Careful state only if the system detected that increasing the sending rate did not produce a higher incoming rate. That is, without the overuse detector reacting. Detected overuse would instead put the system in Decrease mode, followed by a standard Increase. The reason for the difference was to quickly reach the maximum available bit rate. This was designated as the Aggressive build.

The second theory stated that the system should enter the Careful state if any kind of congestion occurred. This could potentially lead to lower bandwidth usage in some cases, but should reduce delay during increase attempts. This build was designated as the Careful build. Some performance differences between these two are shown in the results section.

Suitable threshold values to decide when an increase is successful or for how long the incoming rate must be stable varies depending on the network environment. Extensive testing in several different environments would be necessary to find values that work in all cases. Optimal values may even not exist.

4.3.4 Receiver side communication and connection with Chromium

This module is directly connected to the rest of Chromium. It performs several tasks. The first one is to forward incoming RTP packet information to the measurement modules. The design did not suggest this specific approach, but it made implementation simpler and loosened the coupling between my framework and Chromium by having as few connections as possible.

The second task is to provide a recommended rate that can be sent to the receiver when the application requests it.

The third task is to make sure that the recommended rate is up to date by continuously request rate updates from the measurement modules. At every request, the module rates are compared to find the lowest one. If that rate is higher than the current rate, the system assumes that no module has detected congestion and will increase the sending rate. If the rate is lower than the current one, the system backs down to that rate. It will then hold for a while to see if conditions improve before allowing further rate reduction. This behavior is shared with the delay module, but this module is considering input from several sources that may detect the same congestion event.

4.3.5 Sender side feedback module

This component receives feedback from the receiver side adaptation over RTCP packets. Besides forwarding the bit rate to the video encoder, it also monitors the frequency of incoming reports. It uses the currently measured round-trip time and a known minimum report frequency to decide if the receiver has stopped responding. If this happens, the system is designed so that it reverts to using a minimum video bit rate until feedback is resumed.

5 Test setup

This section details the test setup and test cases. Three Chromium builds were used during the tests. The Careful and Aggressive versions of the test build described in section 4.3.3 as well as one with the standard Google algorithm.

5.1 Test network

The test bed consisted of two laptops with similar specifications acting as clients. The laptops were a Fujitsu Lifebook A series (Intel Core i7-2620M, 4 GB RAM, Windows 7 64-bit) and Lenovo Z570 Ideapad (Intel Core i5-2430M, 6 GB RAM, Windows 7 64-bit). Even though the video communication is done both ways, the Fujitsu laptop was designated as sender and the Lenovo laptop was designated as receiver. The clients will be referred to as sender and receiver from now on. To obtain video, two cameras of model Logitech HD Pro 920 were used.

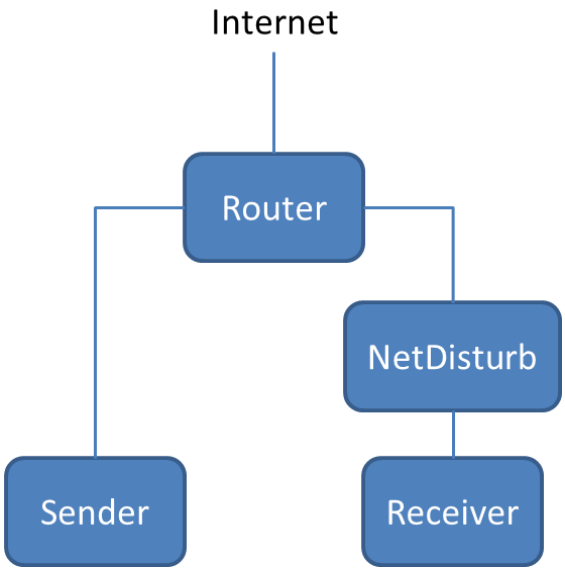

Figure 5 This figure illustrates how the test network was set up. The boxes represent computers and the router, while the lines represent TP cables.

These two clients were connected using a Dlink DIR-655 router. In order to be able to create network impairments at will, a third machine was used. A stationary computer with two Network Interface Cards was placed between the router and the receiver. The stationary computer was running a network impairment program called NetDisturb [37]. This program lets the user create a wide variety of network impairments such as packet losses, delay, jitter and bottlenecks.

5.2 Test web site

During session setup, both clients visited a web site [38] built with Google App Engine and hosted on Google Appspot. The web site has two purposes. First, it provides the HTML 5 web page used to display the video streams. Second, it is required to allow the clients to setup the peer-to-peer connection. This web site was a copy of the official Google WebRTC example page [39]. As the WebRTC implementations in most browsers are still changing, a web page that is not being changed by someone else would be necessary as the same Chromium build would be used during the entire thesis. Previous experience with Chromium has shown that such changes do occur.

5.3 Test cases

The primary test cases consisted of running a video session in four scenarios in addition to a reference case with no impairments. The test cases were designed to be as clear and simple as possible. During the tests, the video stream shown at the sender was compared with the same stream shown at the sender side to get an idea of the video QoE.

The intention was to compare the implementation presented in this thesis with Google's implementation to see how they perform. In both of the implementations the maximum output bitrate is set to 2 Mbps.

In all cases, network impairments were only applied in the sender-receiver direction. The purpose of this limitation is to focus on detection by analyzing the incoming video stream. In a live network environment, congestion can of course appear in both up and down links and in such a case also affect the feedback RTCP messages. There is also the case with LTE, where the downlink traffic can affect the uplink. As all tests presented in the thesis were performed in a local test network with impairments created by a dedicated device, this was considered less of an issue and testing with impairments in both directions was left for future work.

As the most focus of this implementation was put on delay measurement, the buffer size in NetDisturb was set to unlimited to simulate a network with large buffers. In practice, this queue is not unlimited, but simply very large. The developers of NetDisturb have however not specified the maximum size, but even in extremely unrealistic cases this limit was never reached. Another motivation for this choice was that the intention of the rate adaptation systems is to avoid congestion induced packet loss and keep the delay as low as possible. Picking an arbitrary buffer size that would suddenly force reaction based on packet losses instead of delay seemed like a poor solution. This setup would make it very clear if some situations would cause long delays or if they could be avoided.

Something that became apparent after some usage of NetDisturb is that it counted one kilobit as 1024 bits. The extra throughput due to this was in the range of 100 Kbps and is thus small enough to not make a large difference. It should however be taken into account when repeating these tests using a different network impairment software. The rate adaptation systems use the convention that one kilobit is 1000 bits and all plots generated from logging output from the browser will follow this convention.

5.3.1 Static bottleneck

During this test, a static bottleneck of 1.5 Mbps was applied to the network. The purpose of this case was to see how the systems perform in networks with limited bandwidth.

5.3.2 Temporary bottleneck

In this case, the bottleneck is initially set to 2.5 Mbps. The systems are allowed to reach maximum bit rate before a lower bottleneck of 1.5 Mbps is applied for 60 seconds. The purpose of the test is to evaluate how the systems react to a sudden reduction in available rate. Also to see how quickly the systems can recover when the bottleneck is removed. Initial testing showed that both adaptation systems reacted equally fast. Focus of the tests was thus put on the reaction to the bottleneck and the subsequent behavior.

5.3.3 Temporary TCP traffic

In this case the bottleneck is set to 2.5 Mbps during the entire test. The intention of the higher bottleneck was to allow the video stream to run at maximum rate before the TCP traffic was introduced. This would reveal if the rate adaptation systems would refuse to give up any bandwidth and let the TCP stream use the remaining capacity or if they would lower the sending rates. The TCP traffic was created by opening a specific image file of 9.8 MB located on a remote server using Firefox (version 20.0.1) on the receiving client. During a reference test where the TCP stream was alone in the network, it filled up the entire 2.5 Mbps bottleneck. The case was intended to show a basic case of competing traffic caused by the user in a limited bandwidth scenario.

5.3.4 Competition between two video streams

This test case intends to observe the self-fairness of the algorithms. The test network is limited at 2.5 Mbps as in the other cases. One video session is started and is allowed to reach maximum rate. After that, a second stream between the same clients is initiated using a separate instance of Chromium. There is not enough room for both of the streams to run at maximum bitrate without causing congestion. The expected optimal behavior of the two streams is to equally share the available bandwidth of the simulated link.

This case can be considered the most difficult one to solve. Google’s algorithm has been shown to have difficulties here. Due to time constraints, this test was not as thoroughly tested as the others.

5.4 Metrics

The two main factors that are used to compare the algorithms are bandwidth utilization and delay.

The bandwidth utilization is visualized by showing the output bit rate from the rate adaptation algorithms together with the estimated incoming rate at the receiver.

The delay measurement is obtained by using a round-trip time measurement already present in Chromium. This measurement was tested with a series of delay settings in NetDisturb and was shown to be accurate. As impairments were only applied in one direction, the transport delay going in the unimpaired direction was negligible. The Chromium based round-trip time measurement was an accurate indicator of the communication delay and could be used no matter what adaptation system was being used. The output from the delay-based overuse detector is also shown. It displays the mean variance of the inter-frame delay together with a positive and negative threshold.

6 Results

This section will discuss and present some of the results of the implementation proposed in this thesis. It shows plots from a series of test cases. Each test is made with the two versions of my build, Google's build and where applicable, a build that performs no adaptation at all.

6.1 Figure explanations

This section contains explanations of the main three types of plots used in this chapter.

6.1.1 Bit rate plots

The *output bit rate* is calculated by the adaptation systems and sent to the video encoder.

The *Incoming bit rate* is estimated by the delay based adaptation system.

6.1.2 Round-trip time

RTT is the round trip time in milliseconds as measured by the Chromium media transport system. This measurement is used in all test cases with all builds.

6.1.3 Overuse detector

The *Mean variance* is the measured mean variance of the inter-arrival delay as calculated by the simpler delay based overuse detector.

The *Positive* and *Negative thresholds* indicates where overuse and underuse is declared. The thresholds are adjusted based on the mean variance.

The *T* value is used in Google's algorithm and compared with the *Threshold* to detect overuse.

6.2 Reference case

In this case, the video streams are run on an uncongested network. The intention is to show the behavior when they are allowed to reach the maximum sending rate.

6.2.1 Careful/Aggressive

The two versions share the same startup behavior.

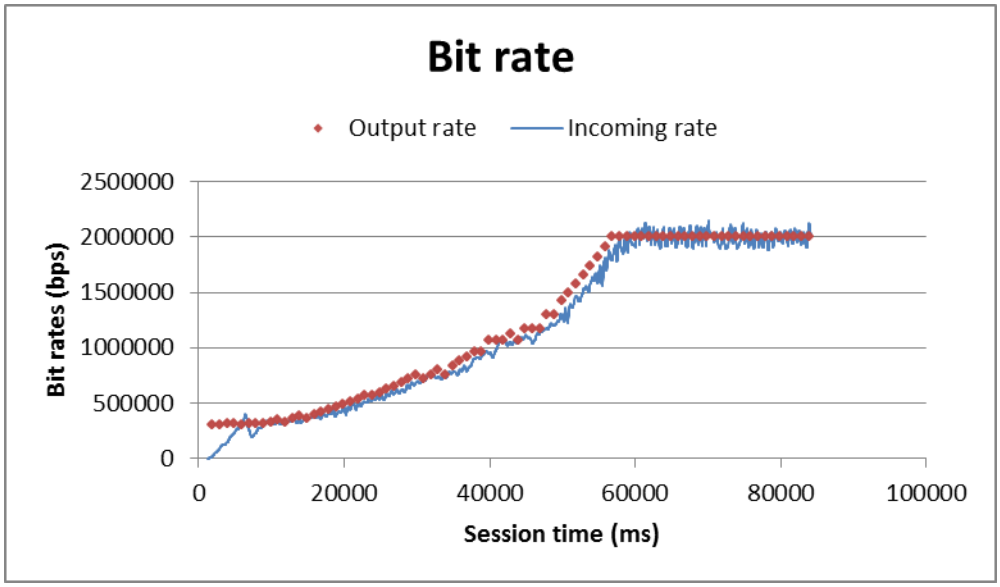

Figure 6 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

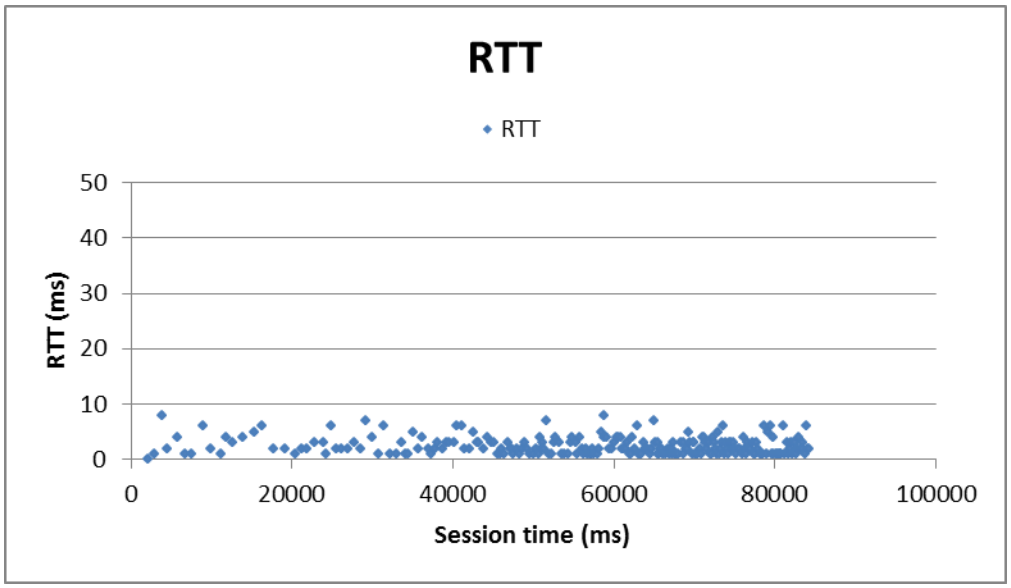

Figure 7 Plot showing the RTT measured at the receiver side during the session.

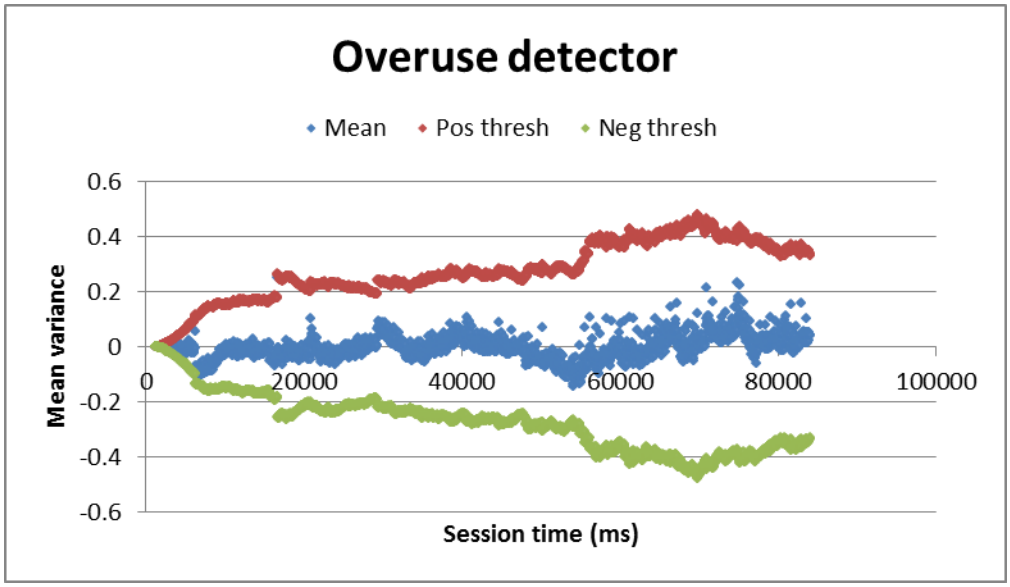

Figure 8 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.2.2 Google

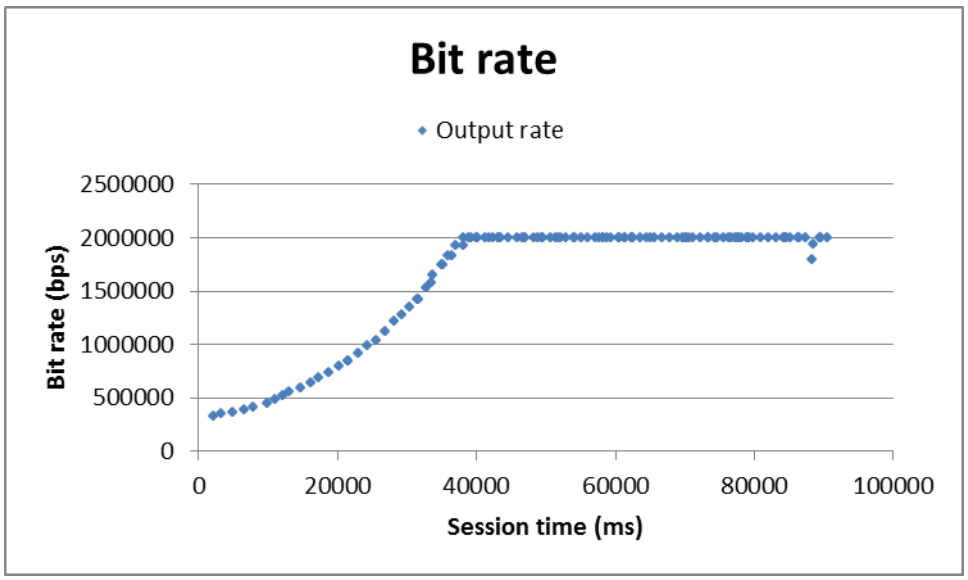

Figure 9 Plot showing the output bit rate of the Google algorithm.

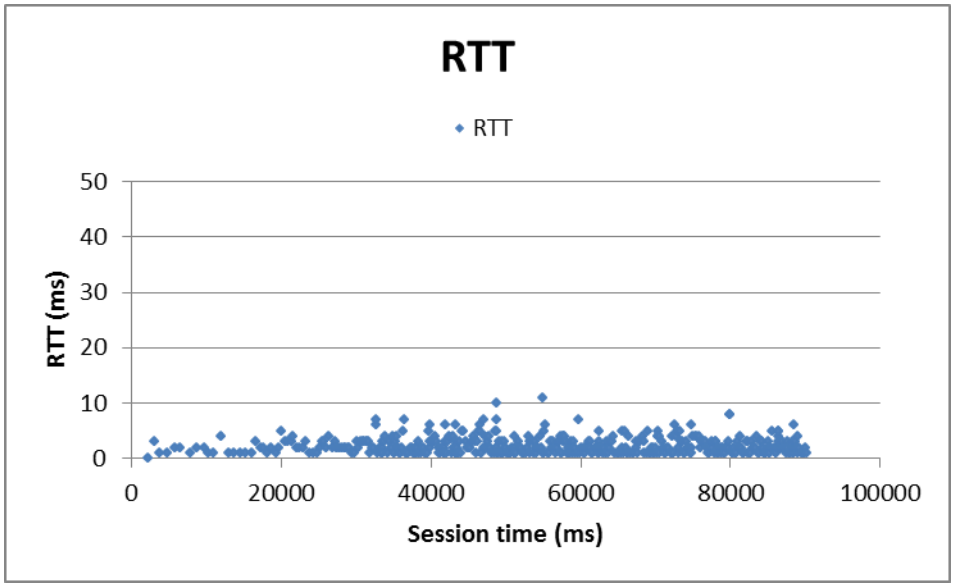

Figure 10 Plot showing the RTT measured at the receiver side during the session.

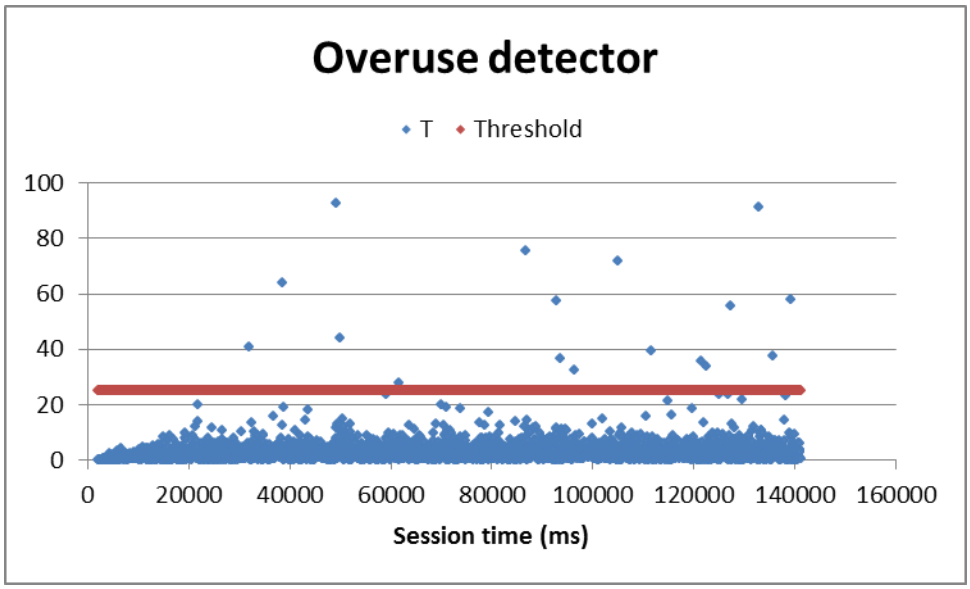

Figure 11 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

6.2.3 Analysis

As can be seen in the plots, both implementations use a slow-start approach. The time required to reach the maximum rate varies between 40 to 60 seconds. The sending rate is steadily increased until the maximum rate is reached. The measured round-trip time is low in both cases. The one-way delay is half of that in each direction. This indicates that the delay is very low on an uncongested link. This should be kept in mind when reviewing the results from the other test cases where the delay in the congested direction will eclipse the delay in the other direction.

The overuse detector plot shows that the mean variance stays well within the limits of the thresholds. The mean variance and the thresholds are all very near zero, indicating a stable system.

6.3 Static bottleneck

6.3.1 Careful

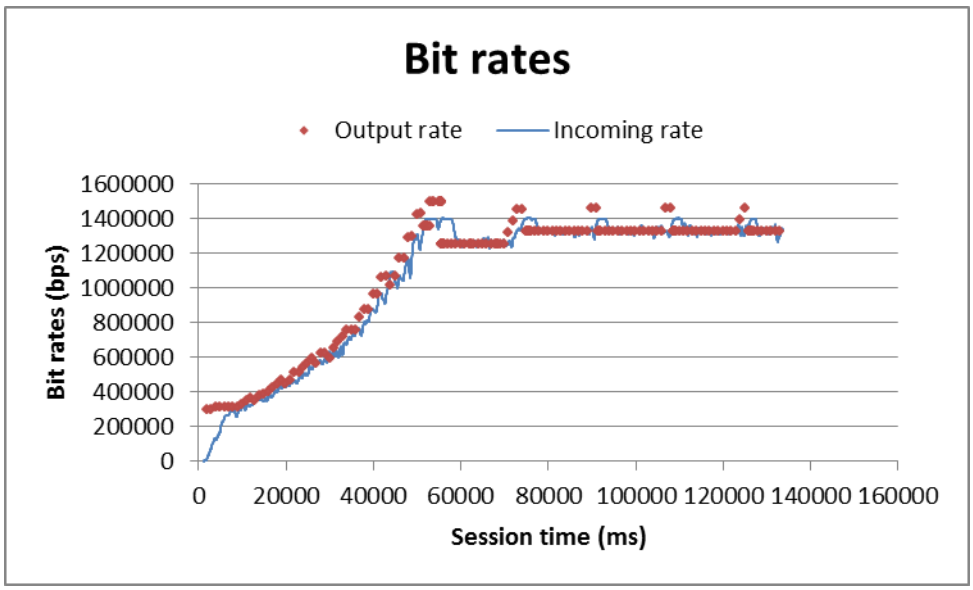

Figure 12 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

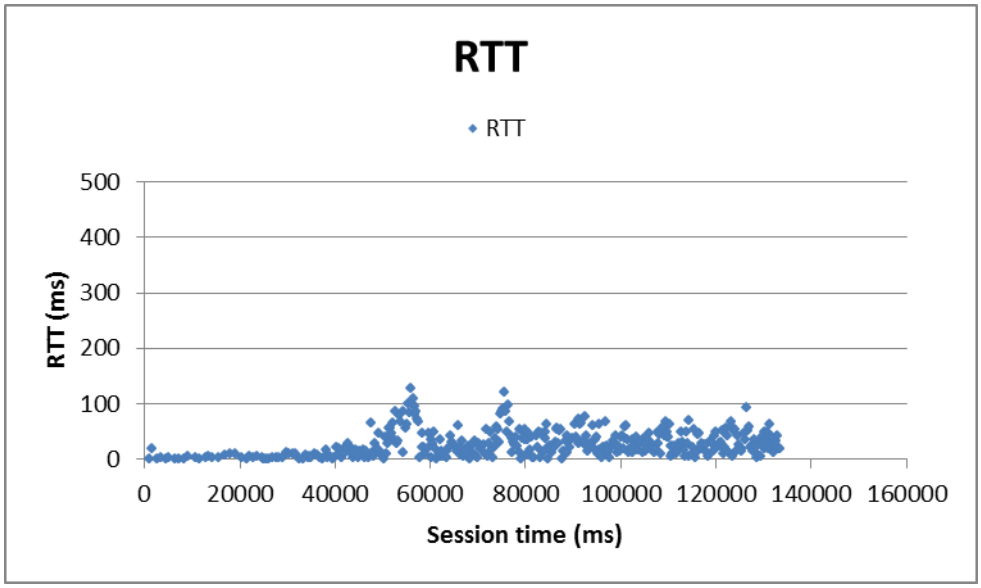

Figure 13 Plot showing the RTT measured at the receiver side during the session.

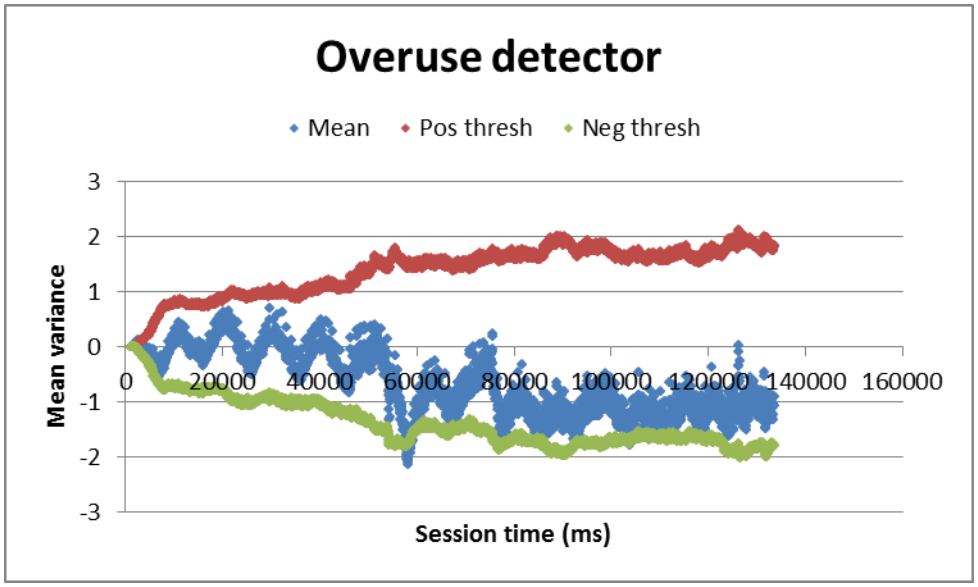

Figure 14 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.3.2 Aggressive

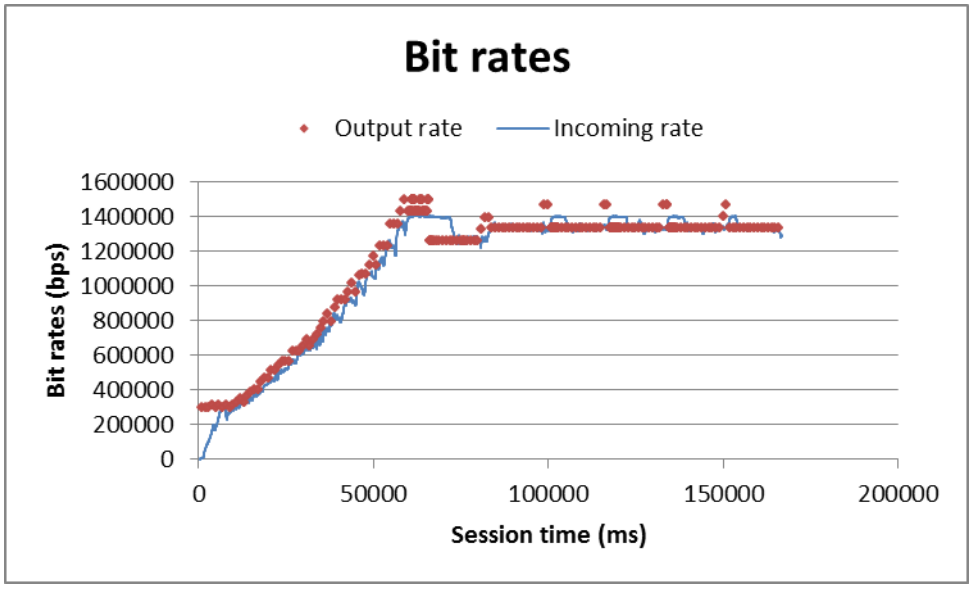

Figure 15 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

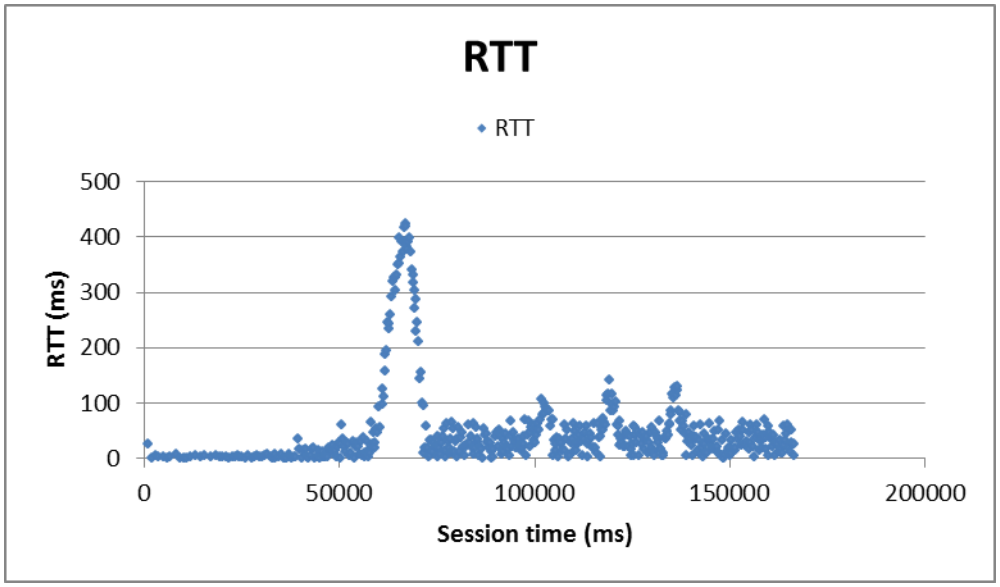

Figure 16 Plot showing the RTT measured at the receiver side during the session.

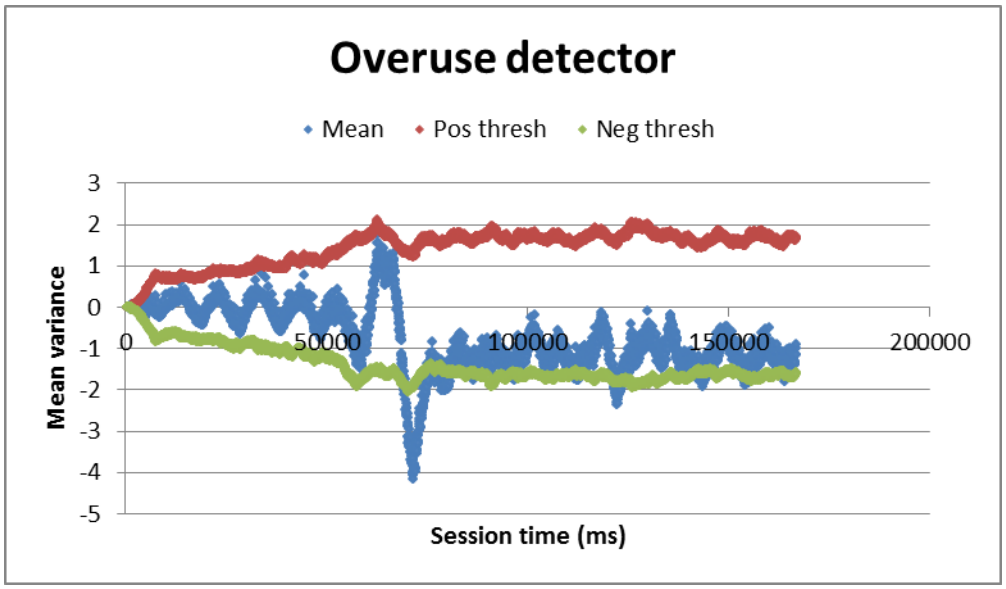

Figure 17 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.3.3 Google

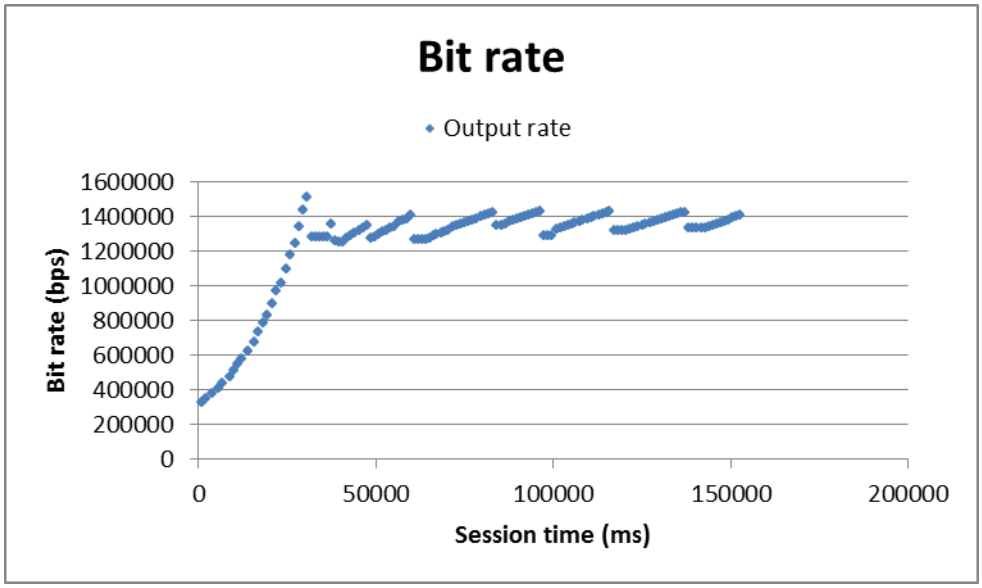

Figure 18 Plot showing the output bit rate of the Google algorithm.

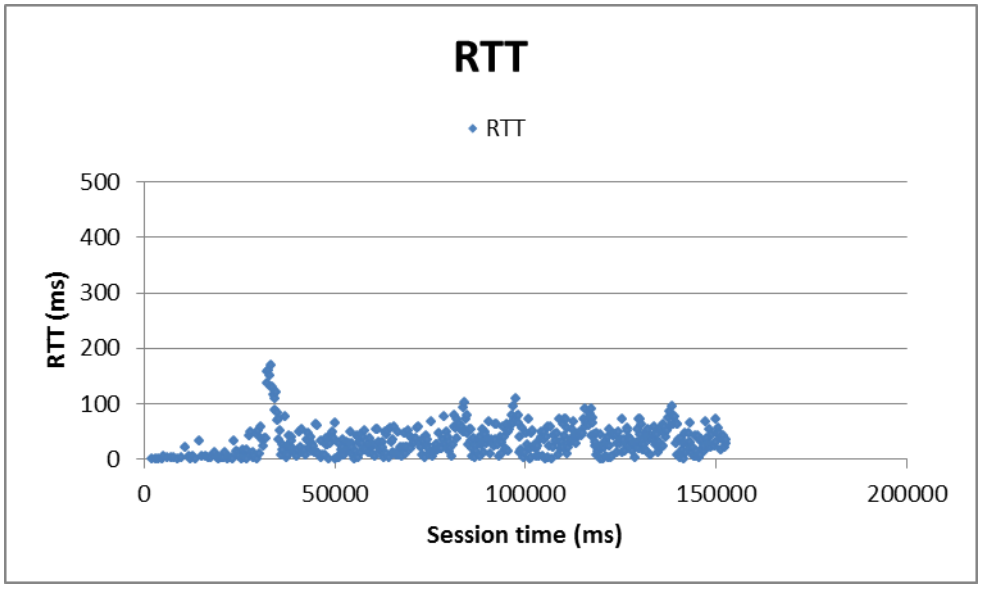

Figure 19 Plot showing the RTT measured at the receiver side during the session.

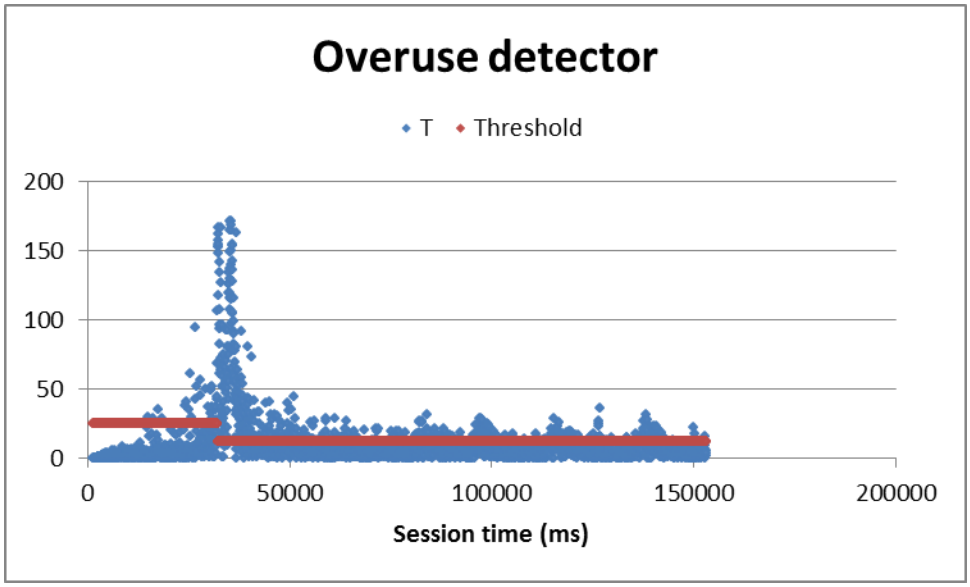

Figure 20 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

6.3.4 Analysis

The aggressive build is able to maintain a slightly higher average bit rate than the careful one. The drawback is that staying closer to the limit also causes a slightly higher RTT. Both builds can be seen staying at a rate slightly below the available capacity while attempting increases from time to time. In comparison, the Google system uses a slightly different approach where it slowly increases the sending rate before dropping it upon overuse. Both test builds and Google's algorithm continuously attempt to increase the sending rate in two slightly different ways.

Both ways are viable as they do not increase the measured delay by much. It can be discussed how much the sending rate should be lowered upon detected congestion.

The difference in overuse detection can be seen by comparing Figure 14 and Figure 20. Google's solution uses a relatively stable threshold as compared to the test implementation.

Some slight video stutter was visible at the point where the Google build and the Careful build reached the bottleneck. In some cases, as can be seen in the Aggressive plots, an initial peak in delay can occur if adaptation is not triggered fast enough. This behavior was occasionally observed with all three builds and shows the importance of reacting quickly and not overshooting the available bandwidth. The aggressive build took longer time to react, which caused a larger RTT peak. Something worth observing is the overuse detector plots. In neither of the two cases is the overuse detector able to react. It is close in the Aggressive plot.

Both systems enter the careful state as can be seen in the 15 second holding periods followed by attempts to increase the bit rate.

6.4 Temporary bottleneck

6.4.1 Careful

The temporary bottleneck is applied after 62 seconds.

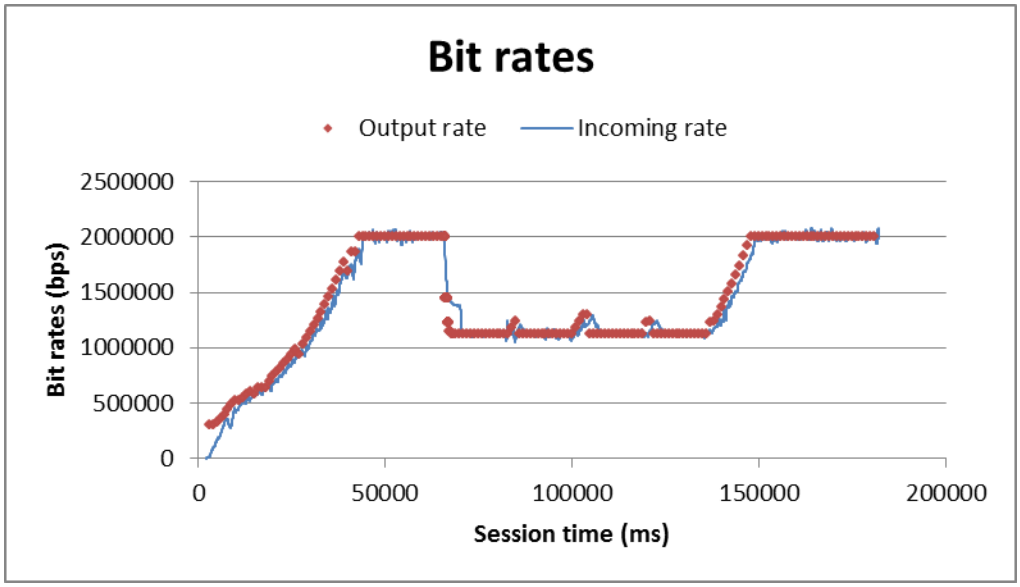

Figure 21 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

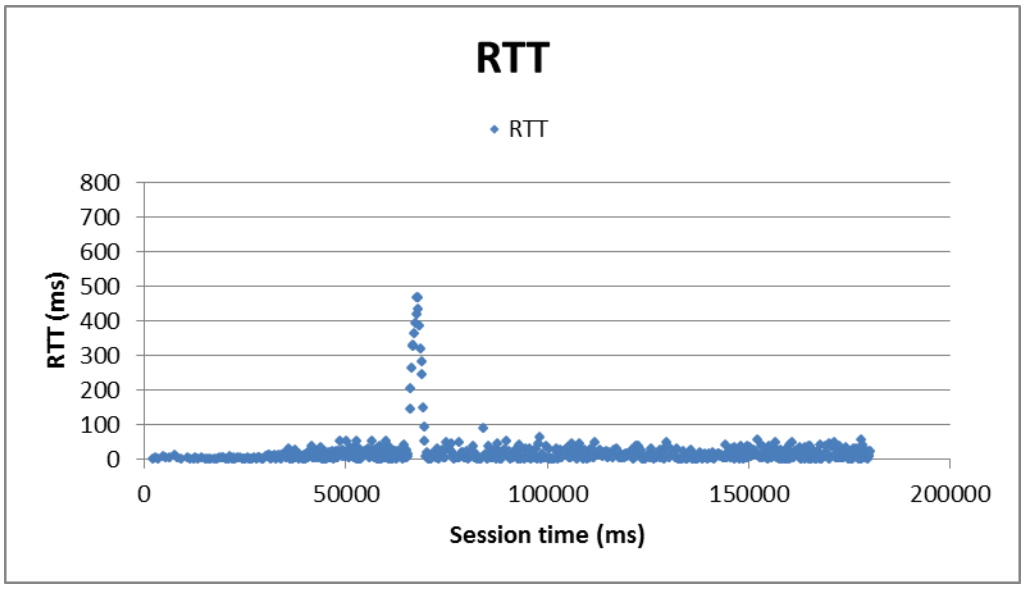

Figure 22 Plot showing the RTT measured at the receiver side during the session.

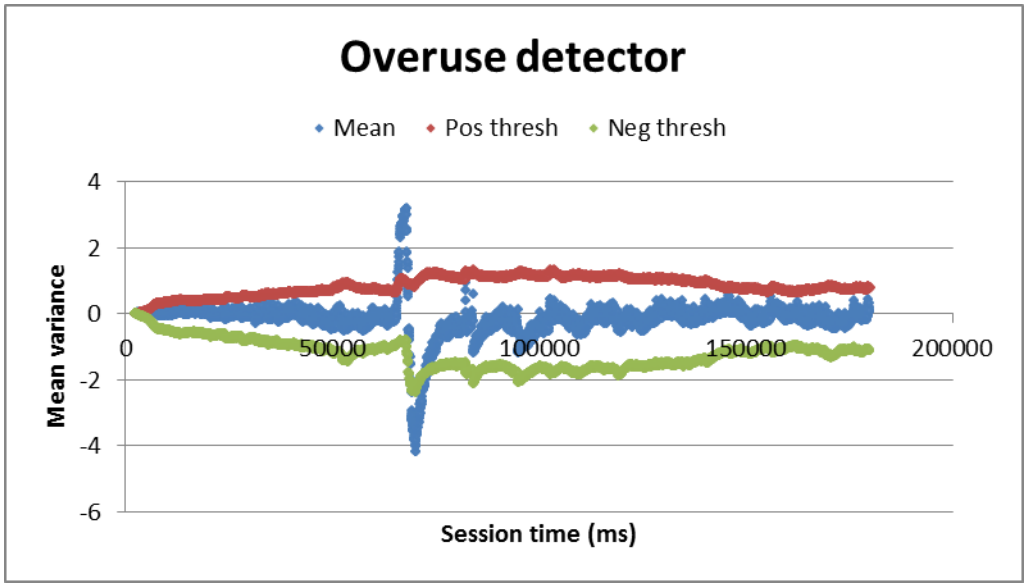

Figure 23 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.4.2 Aggressive

The temporary bottleneck is applied at 75 seconds.

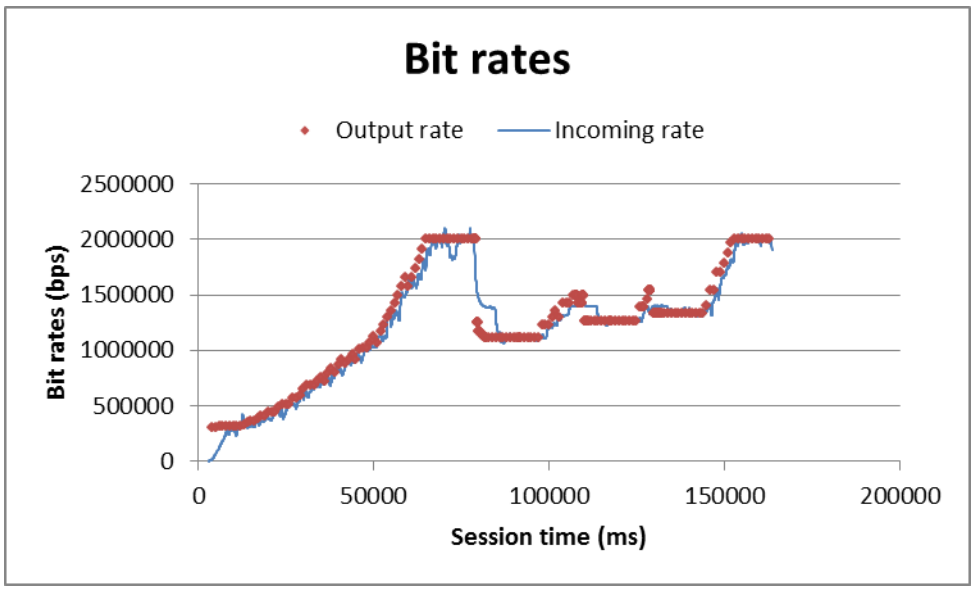

Figure 24 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

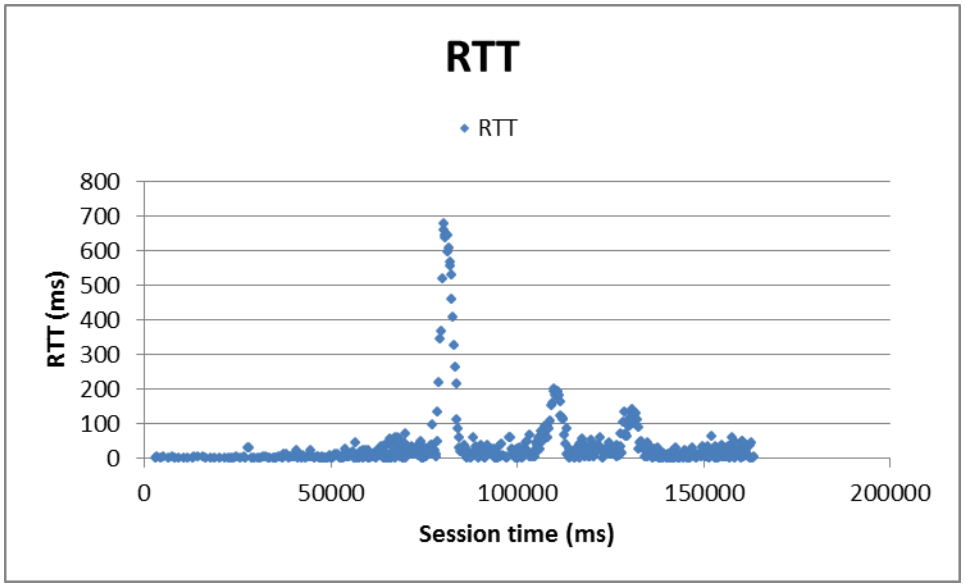

Figure 25 Plot showing the RTT measured at the receiver side during the session.

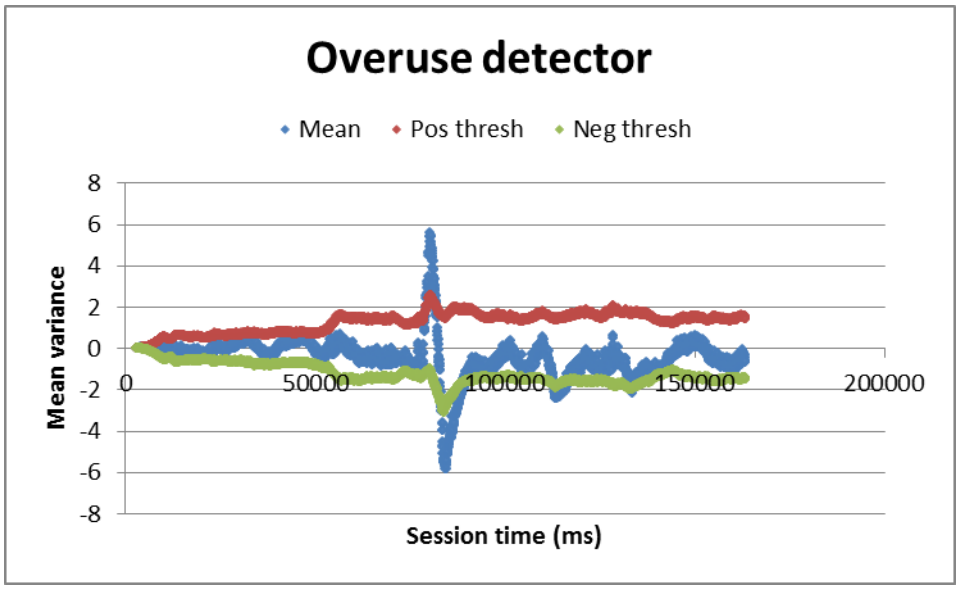

Figure 26 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.4.3 Google

The temporary bottleneck is applied after 65 seconds.

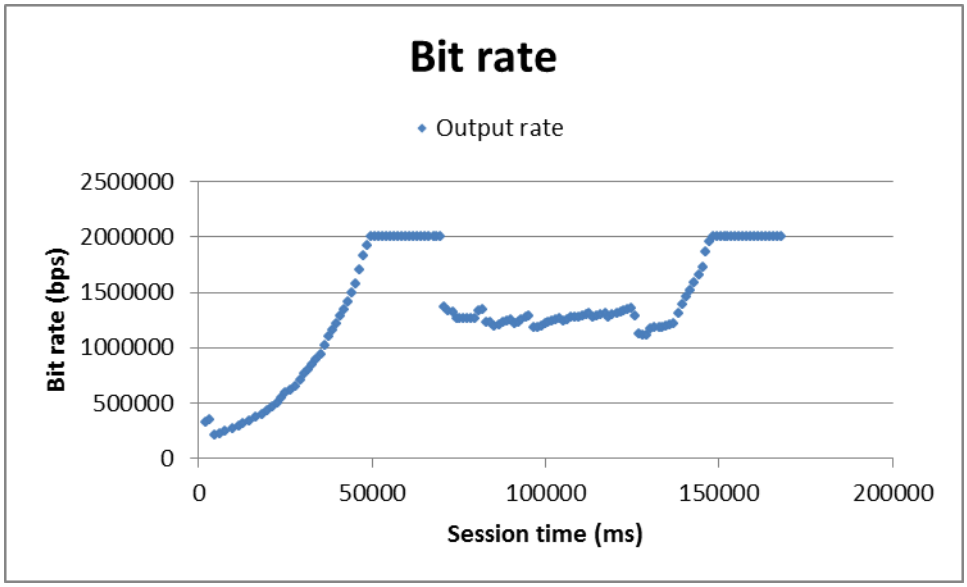

Figure 27 Plot showing the output bit rate of the Google algorithm.

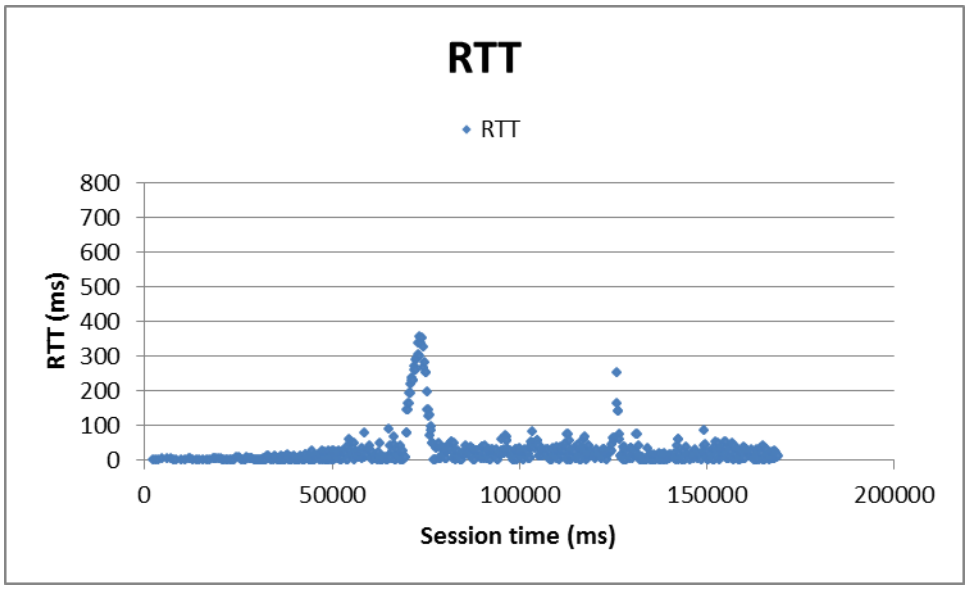

Figure 28 Plot showing the RTT measured at the receiver side during the session.

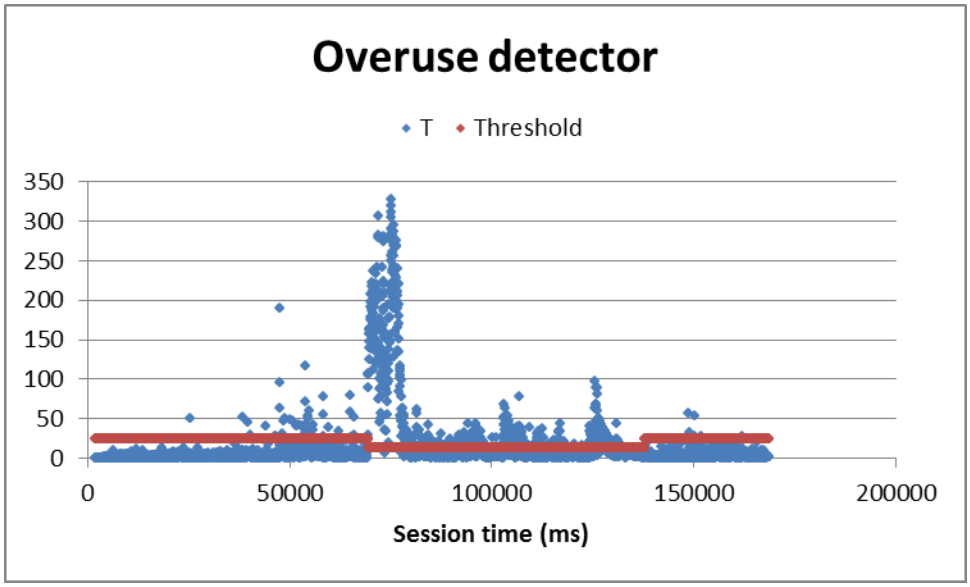

Figure 29 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

6.4.4 Analysis

All tests showed a somewhat quick reaction to the appearance of the bottleneck. The reaction time was still not short enough to prevent the large initial peak in delay. This indicates that it is difficult to react to a sudden reduction in bit rate without any previous warning. In all cases, a one second freeze of the video was followed by a couple of seconds of noticeable delay. This time coincides with the parts of the plot where the estimated incoming bit rate exceeds the target bit rate. The Google algorithm was able to achieve the lowest initial delay peak, but not by much. It was also able to achieve a slightly higher throughput.

After this initial peak, no video delay was visible for the Careful and Google builds. The Aggressive build showed some delay after the initial decrease where it attempts to reach the highest possible sending rate. This behavior lets it obtain a slightly higher rate than the Careful build, but at the cost of a temporary reduction in user experience. On the other hand, staying too close to the perceived maximum available bit rate comes with the risk of slightly overusing the link with a slow buildup of delay as consequence. This test shows a clear difference between the Careful and Aggressive builds.

Fine-tuning the overuse detector thresholds does not make much of a difference in this test case. With the sudden appearance of the bottleneck, all three test builds react immediately.

6.5 Competing TCP traffic

6.5.1 Only TCP

The following plot is generated from a network dump using Wireshark. It shows how the TCP stream behaves when it is alone in the network and may consume as much bandwidth as it wants.

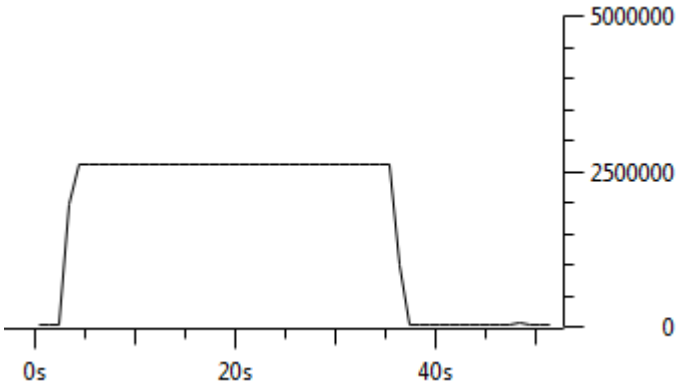

Figure 30 Wireshark plot showing how the TCP stream uses up all available bandwidth.

As can be seen, all of the available capacity is used up.

6.5.2 No adaptation

This plot shows the incoming video stream (red, larger) and the TCP stream (black, smaller). Compared to the plot in the above figure, the image download takes much longer as it is allotted less bandwidth. The delay in the video stream is also very visible as shown in the next figure.

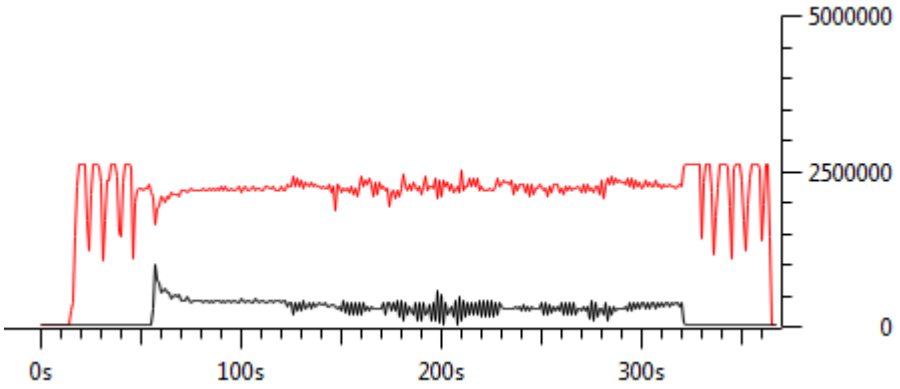

Figure 31 Wireshark plot showing incoming TCP (black) and video (red) at the receiver side.

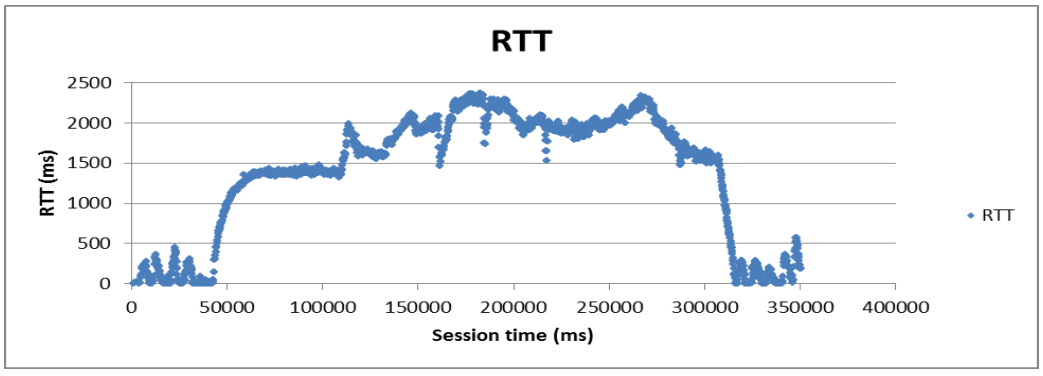

Figure 32 Plot showing the RTT measured at the receiver side during the session.

6.5.3 Careful

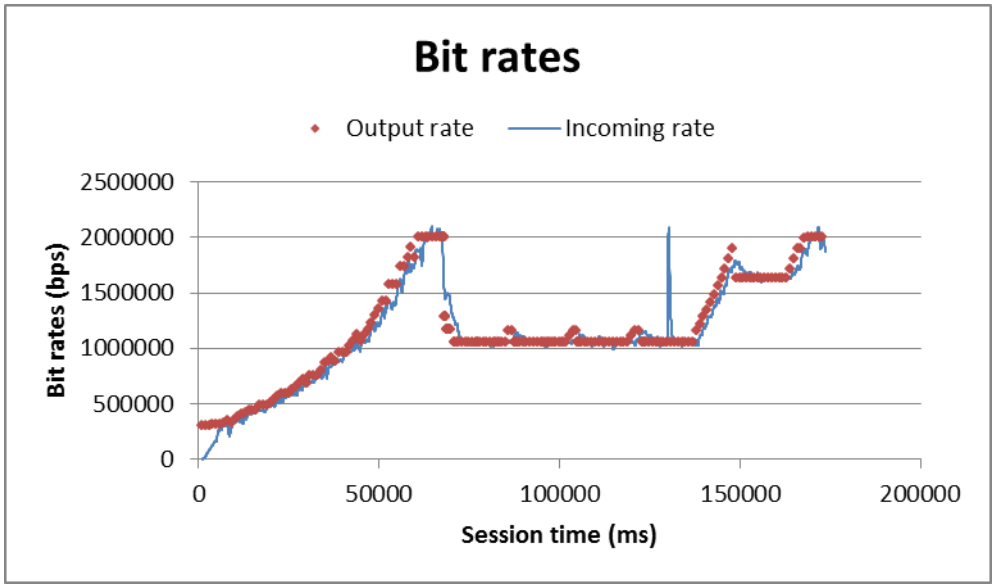

Figure 33 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

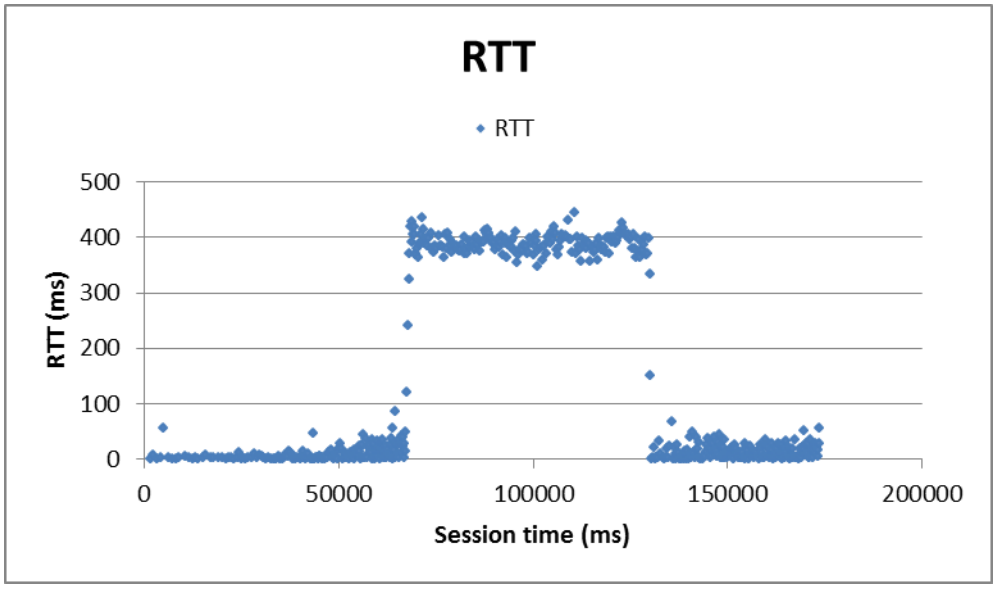

Figure 34 Plot showing the RTT measured at the receiver side during the session.

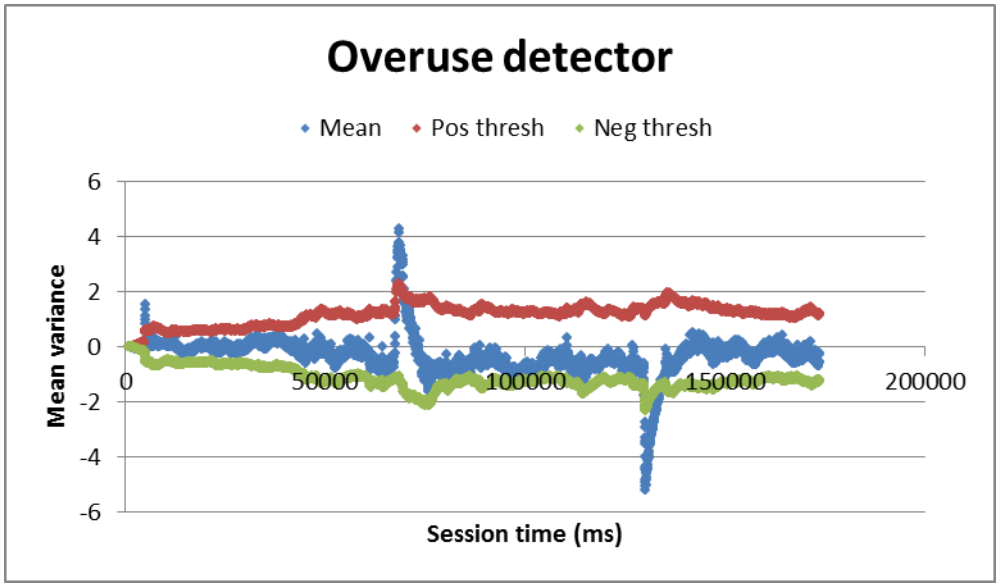

Figure 35 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.5.4 Aggressive

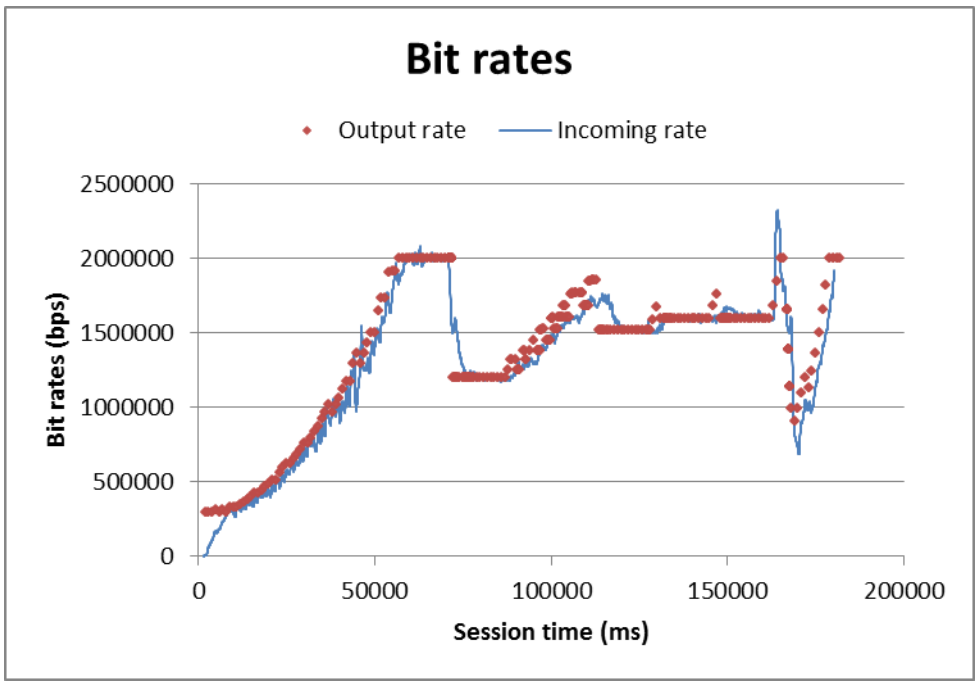

Figure 36 Plot showing the measured incoming bit rate versus the output bit rate calculated by the algorithm.

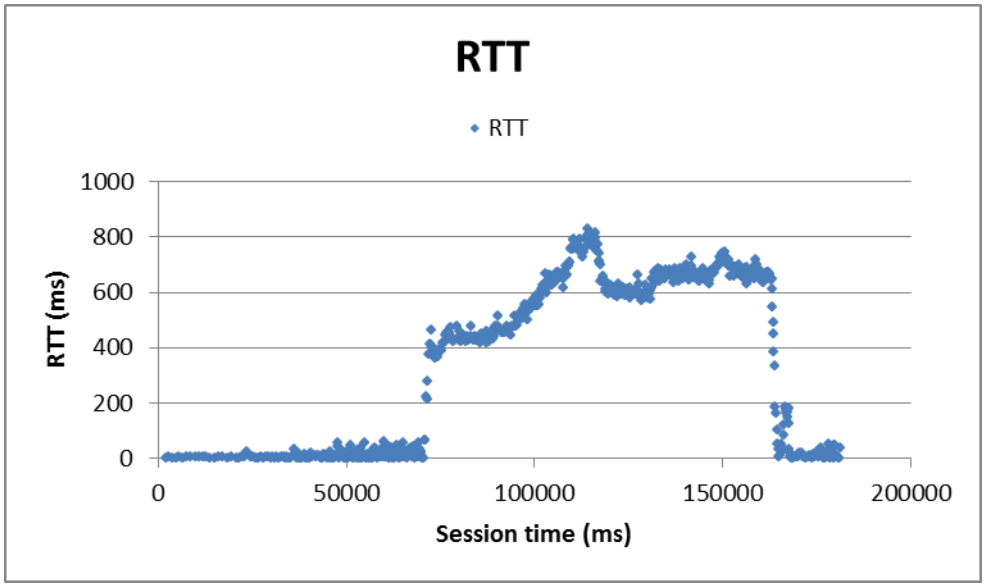

Figure 37 Plot showing the RTT measured at the receiver side during the session.

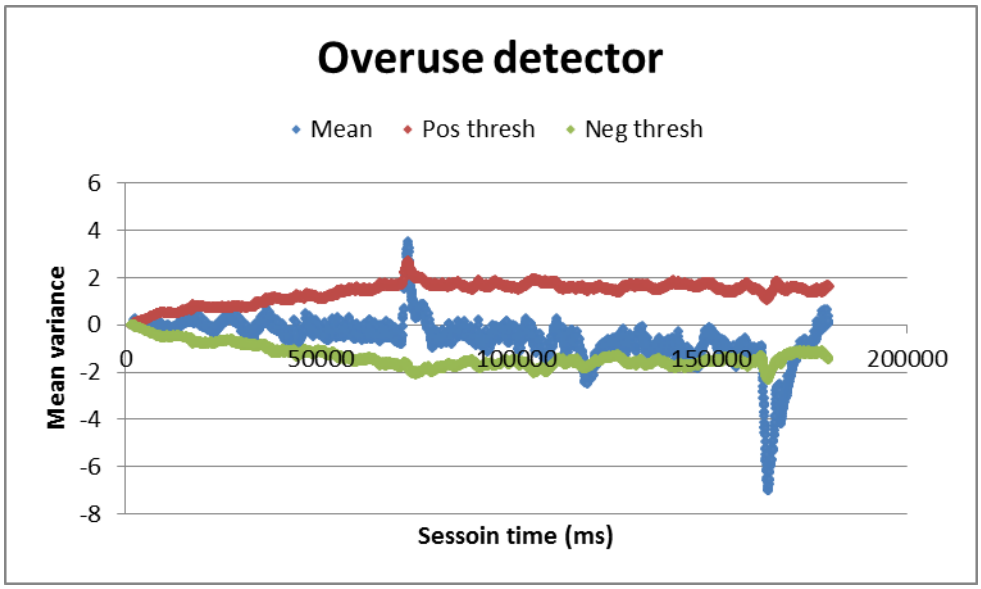

Figure 38 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.5.5 Google

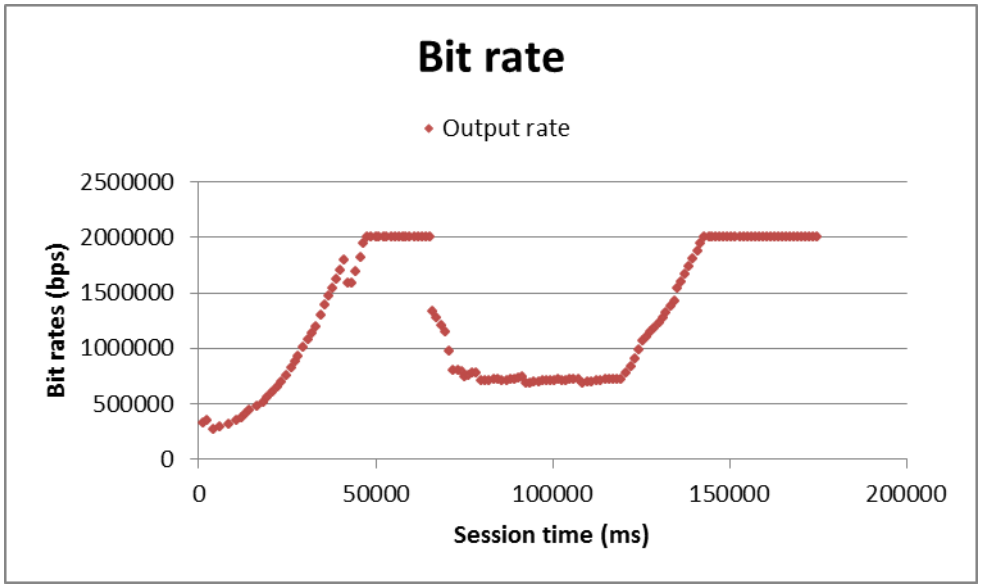

Figure 39 Plot showing the output bit rate of the Google algorithm.

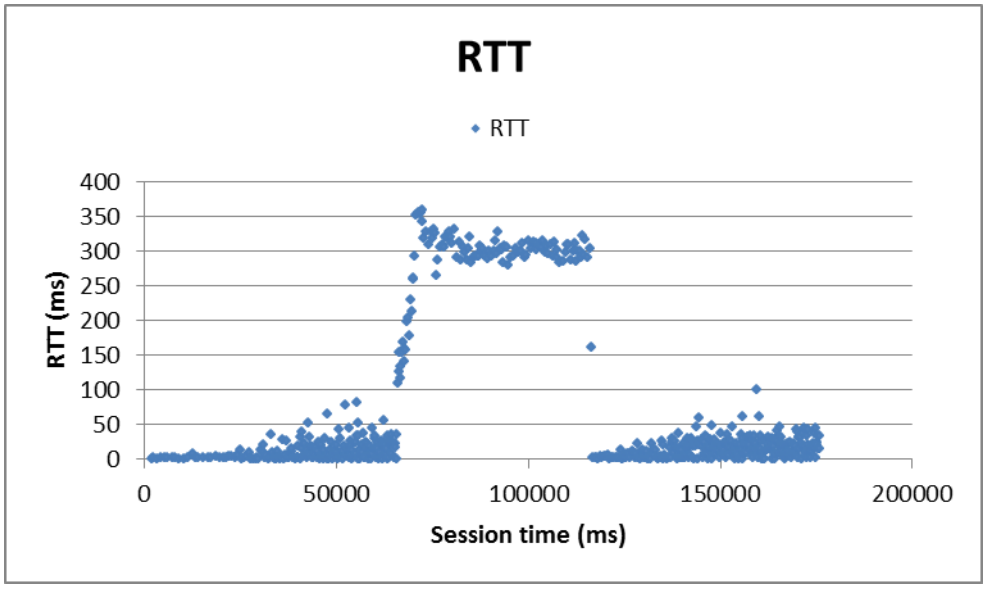

Figure 40 Plot showing the RTT measured at the receiver side during the session.

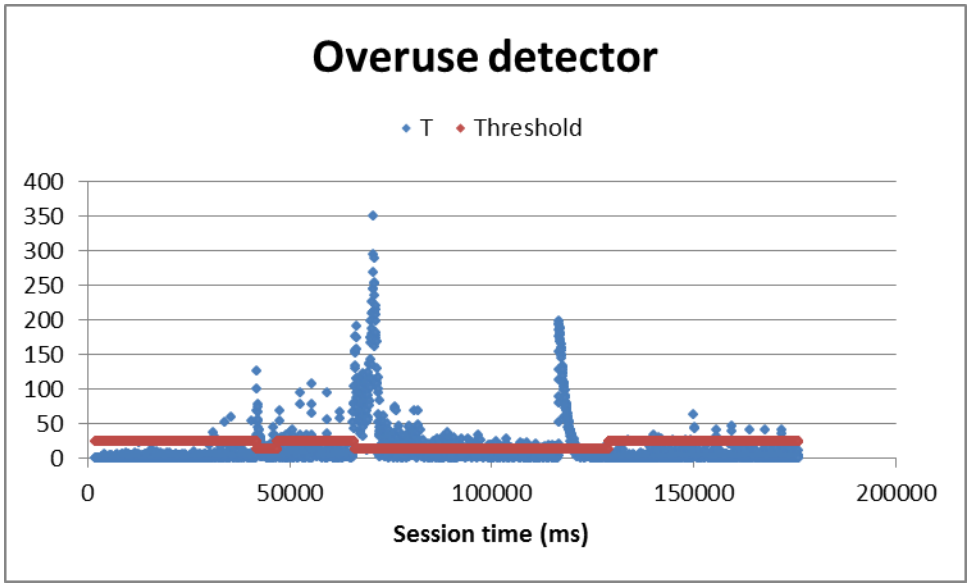

Figure 41 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

6.5.6 Analysis

The results imply that the TCP stream will use any available bandwidth. While the previous test cases involving bottlenecks (sections 6.3 and 6.4) have shown similar behavior, this case gives us three different strategies together with their consequences on delay.

The Aggressive build stands out as it is actively trying to increase the rate after the initial drop. This shows that the browser TCP traffic can be forced out if necessary. This comes at the cost of higher delay and longer duration of the TCP session as the throughput is lower. Test cases show a peak in estimated incoming bit rate after the TCP session has ended. The Careful build and Google's build shows similar behavior. They both drop the rate and remain stable for the duration of the TCP stream. The difference in sending bit rate is around 360 Kbps. The difference in delay is approximately 100 milliseconds as can be seen by comparing Figure 34 and Figure 40. A third difference is the length of the TCP session, as the image file will be downloaded faster if it is given more bandwidth. The results suggest that some delay will be occurring even if the video bit rate is lowered. It is still far better than if the rate is kept constant as can be seen by comparing Figure 34 and 40 with Figure 32. In Figure 32 it is also visible how the browser is performing some additional rate adaptation on top of the standard TCP adaptation as the.

The Aggressive case shows how the packet loss detection module reacts to perceived loss or packet reordering and temporarily forces the rate to go down. This can be seen at around 160 seconds into the session. The effect is temporary as the packet loss module will let the rate increase again as soon as no losses are detected.

The video was still smooth with both the Careful build and the Google build, and the visible delay was constant during the TCP session, as the plots indicate. As long as neither the video stream nor the TCP session attempts to increase the bit rates, the RTT remains stable. This suggests that some delay may have to be accepted, and that rate adaptation systems have to decide between having higher video quality and giving up more bandwidth in the hope that the competing TCP traffic will be short-lived. There is of course no guarantee that the competing traffic will end within a reasonable amount of time. In these cases it may be necessary for the video rate adaptation systems to accept higher delays in order to fight for the available bandwidth.

6.6 Competing video streams

This test case received the least amount of attention due to time constraints. The limited testing that was performed did reveal some interesting things. Tests performed by [49] suggest that the Google algorithm has problems with this case where one stream will starve the other stream. RTT plots for both clients were identical in all test cases, therefore only one RTT plot per build and case is shown.

6.6.1 Careful

These plots show how the careful build reacts when one stream is running at maximum bit rate and another video stream appears.

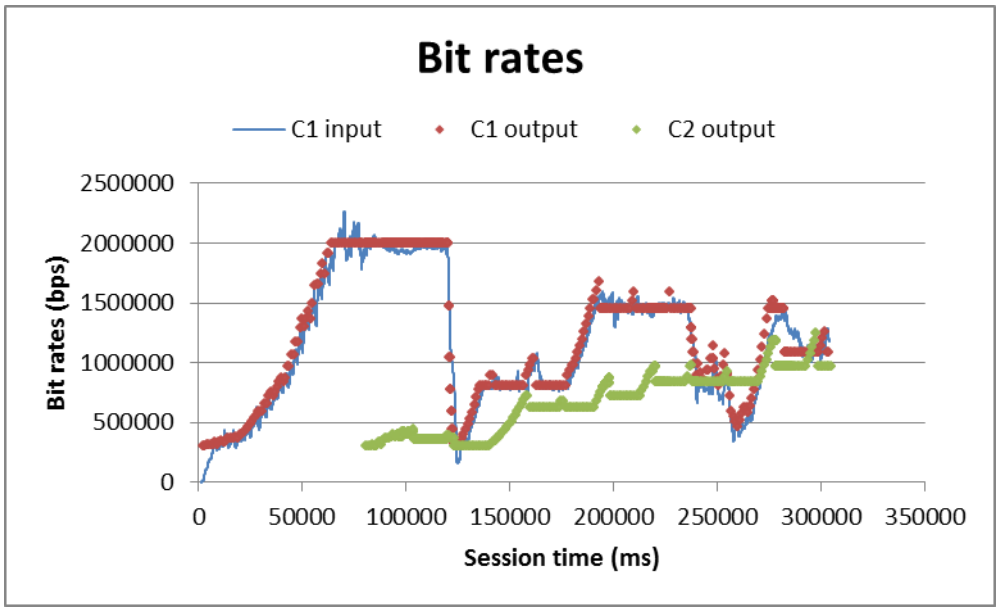

Figure 42 Plot comparing the bit rate of an initial stream with the bit rate of a second stream that appears once the first one has reached its maximum value.

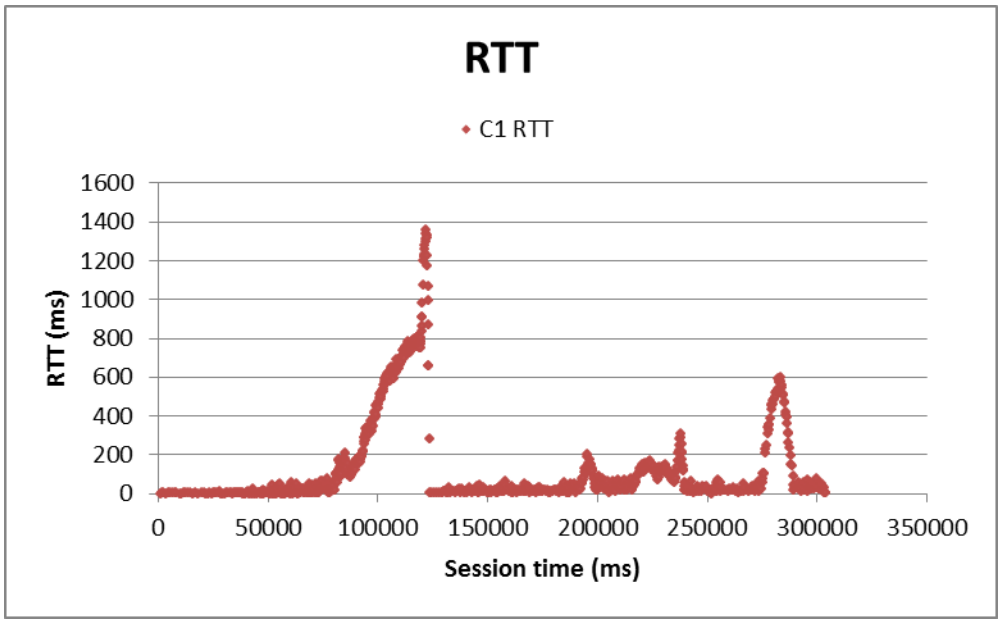

Figure 43 Plot showing the RTT as measured by the Client 1. The RTT of Client 2 matched this plot and is omitted.

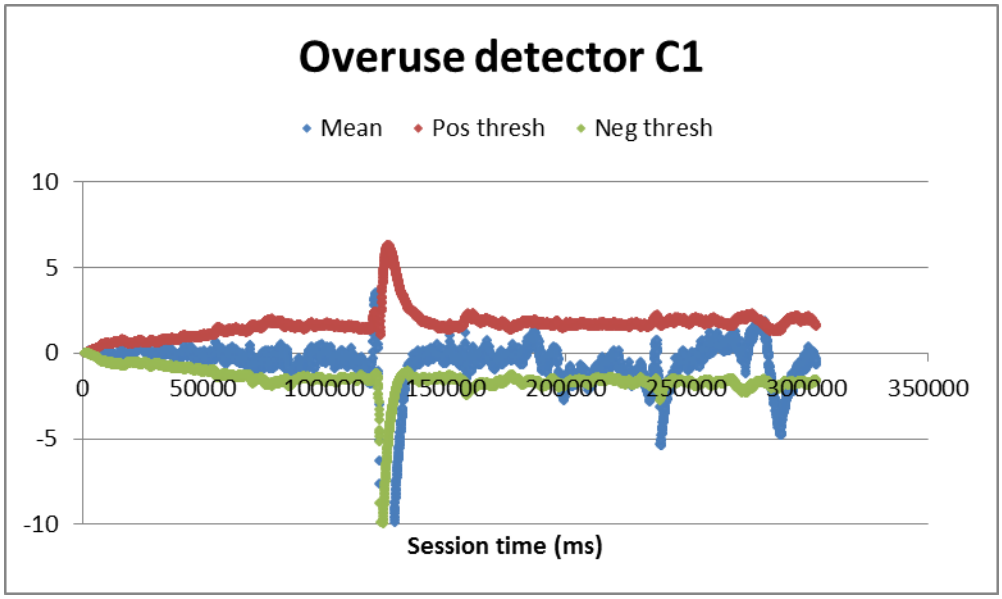

Figure 44 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

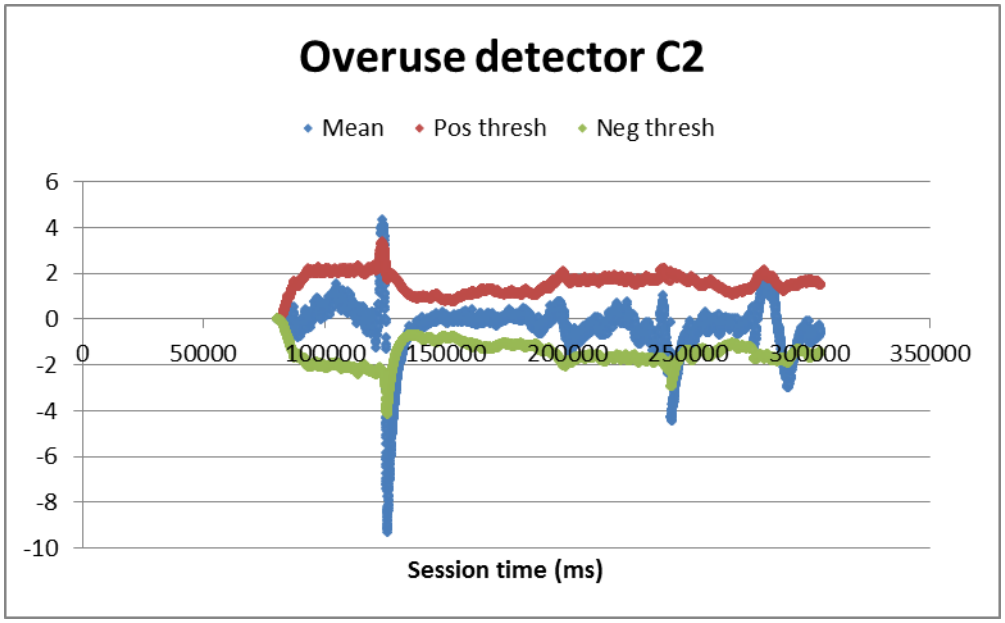

Figure 45 Figure showing the mean variance (middle group) compared to the positive threshold (upper group) and the lower threshold (lower group).

6.6.2 Google

These plots show how Google's adaptation system reacts.

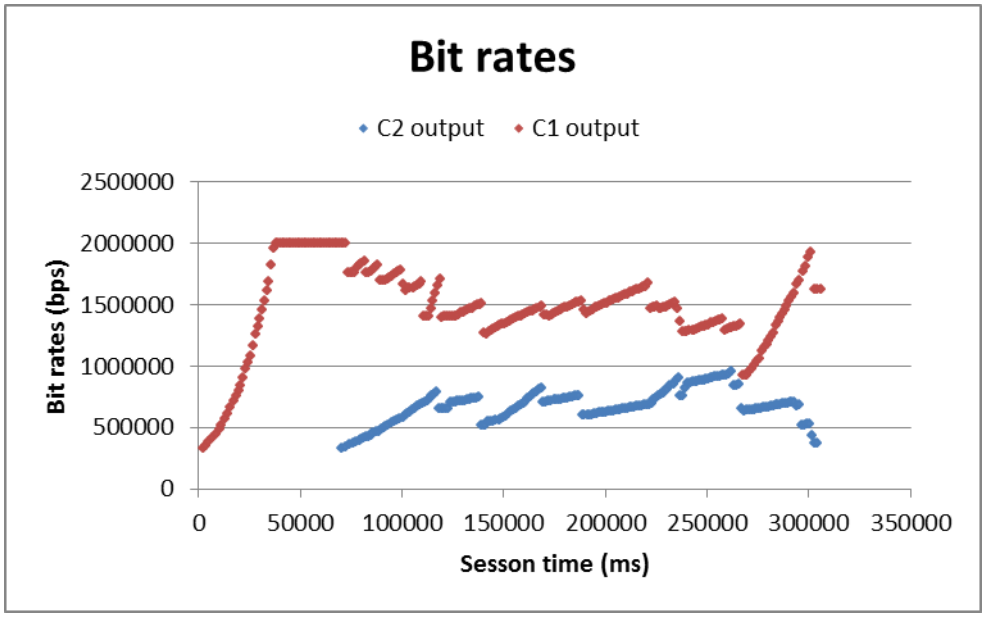

Figure 46 Plot comparing the bit rate of an initial stream with the bit rate of a second stream that appears once the first one has reached its maximum value.

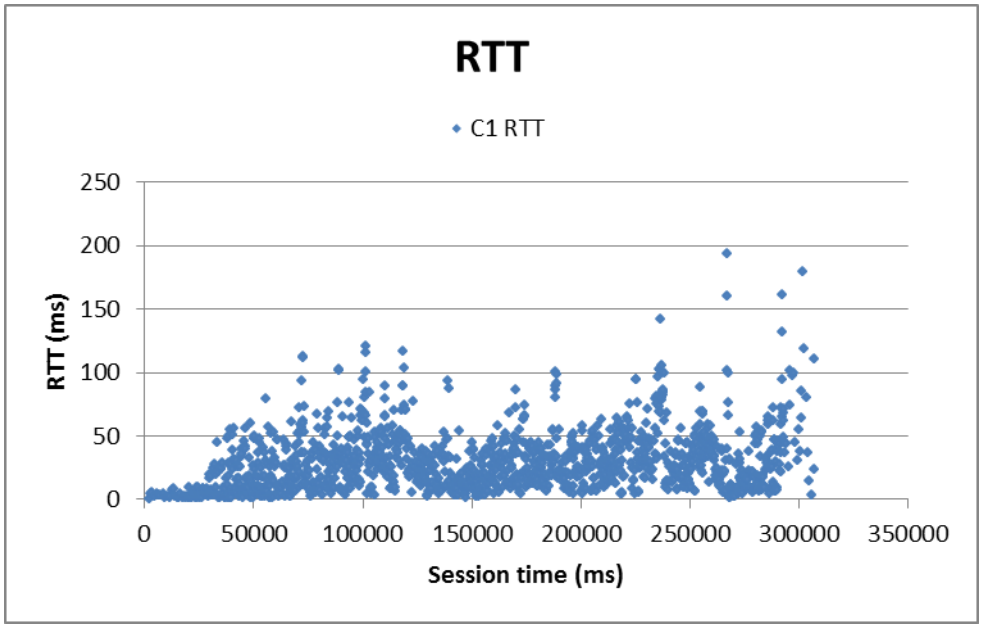

Figure 47 Plot showing the RTT as measured by the Client 1. The RTT of Client 2 matched this plot and is omitted.

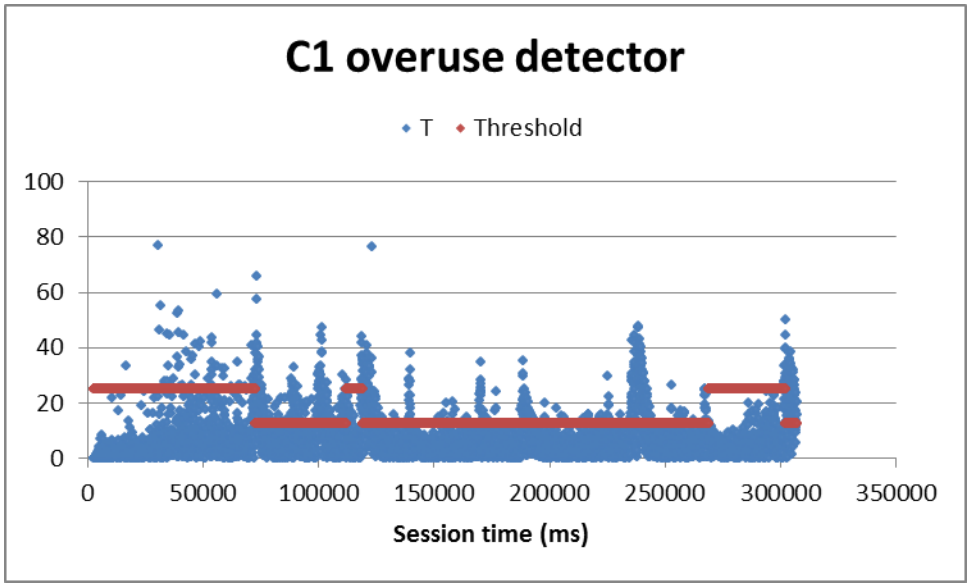

Figure 48 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

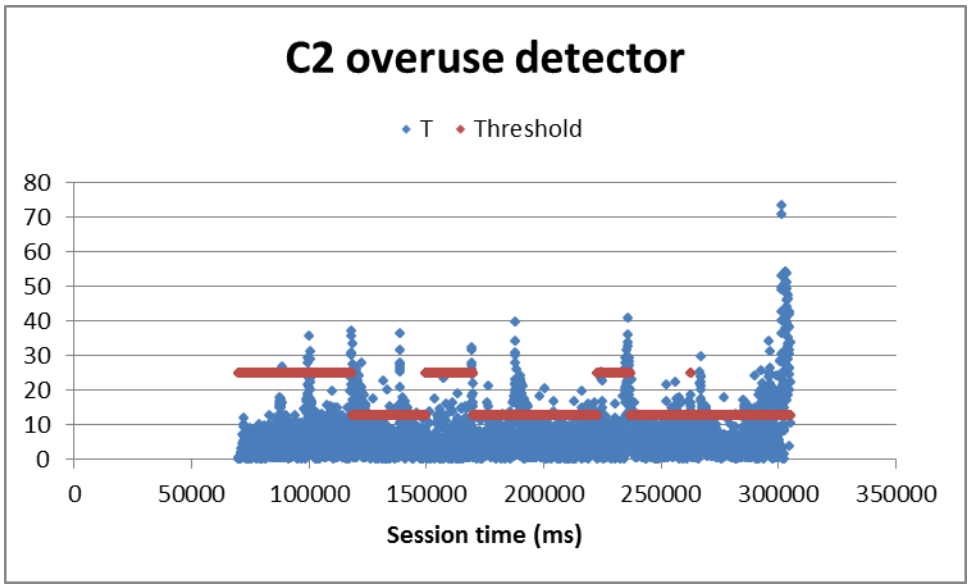

Figure 49 Plot showing the T value of the overuse detector (dots) compared to the current threshold (line).

6.6.3 Analysis

As can be seen in the two cases, none of the systems perform very well. The Google adaptation algorithm is very careful during startup; hence the first stream is unable to grab more bandwidth. The first stream is able to detect a slight increase in delay and lowers its sending rate slightly as a response. Unlike the careful build, the participants of both streams detect congestion and reacts at once, but the response is not fair. Both lower the rate equally, preserving the unfair bit rate ratio.

The case of the careful build reveals a major weakness of the system. This is clearly visible in Figure 43. The second stream fills up the remaining available bandwidth and requires a little bit more. While the second stream is increasing the bit rate, the delays of both streams is increasing slowly. The rate of this increase is low enough so that the overuse detector does not react. To handle this, the system would need additional input to react upon. Once congestion is detected, both streams react by reducing the sending rates to minimum. This behavior results in both streams having to start again and sharing the available bandwidth equally.

In addition to the metrics discussed, the rate adaptation system would benefit from observing the estimated round-trip time in order to detect a slowly increasing network delay. Small continuous increases in frame delay were not noticed by the delay-based overuse detector as it focuses on inter-arrival time. If the time difference between each frame is low enough, the system would not perceive any delay. By observing the one way delay (OWD) one could react to different tendencies such as rate of increase and longer times with high OWD. One could also confirm the usefulness of reducing the sending rate by observing if the OWD is reduced or not. A system could also have set policies that prevent any rate increase if the delay reaches above a certain level.

One problem with handling the self-fairness case seems to be that the first stream needs to be able to detect the new stream as it is increasing in rate. This requires some fine-tuned detection method to detect it if the new stream is increasing its rate slowly. The other side of the problem is that the new stream must dare to fight for increased capacity even if it could lead to a temporary increase in delay. A possible solution would be to use a very careful adaptation technique where the rate is heavily reduced upon detected congestion. This could let the two streams compete over the available capacity from an equal starting point. This strategy could on the other hand cause worse performance in other cases. An example would be the case where other network users do not follow the same strategy and instead increase their bandwidth usage when space is freed up. This remains untested in this thesis, but could be an interesting topic for future studies.

7 Conclusions

In conclusion, the implementation proposed in this thesis is comparable to Google's in the five cases shown. The results indicate that a simpler delay-based overuse detector is comparable to Google's more complicated one. Detection and reaction to bottlenecks works well and introduce only a short time of delay. The two different overuse detector threshold strategies are both working. The Google solution produces a T value that is a bit more stable and can thus have a more stable threshold. The opposite is true for the test implementation. A general conclusion that can be drawn from the testing is that it is very difficult to produce an algorithm that can detect and react to several different cases of network congestion. A solution that works well in one case might be very ineffective in other cases.

Both systems reduce their bit rate when faced with TCP traffic caused by web browsing. A relatively high delay occurs in all cases even if video rate was adapted. Results from running the aggressive version suggest that it is possible to obtain higher bit rates at the cost of higher delay. The tests indicate that one has to accept increased delay any when competing with long lived TCP flow when there is no AQM implemented in the communication path. This is expected since TCP without AQM or ECN typically operates with full buffers at bottleneck links. Even though the streams were delayed, there were no artefacts or other signs of network issues. This indicates that adaptation systems may have to choose between maintaining a high bit rate with longer periods of delay and using a lower bit rate hoping that giving the competing stream more space will let it complete its transmission. While this delay was visible when being able to compare directly with the video source, the video was still smooth; similar to if the delay was due to a very large distance between the two parties.

A final conclusion is that detecting and reacting to congestion using a delay based approach turned out to be the easy part. Deciding how to react was more difficult.

8 Discussion

8.1 Implications of the conclusions

The development of congestion detection and rate adaptation systems have moved from initially reacting to the effects of existing congestion to attempting to reduce the congestion before the effects become noticeable. There is still a limit to how much that can be done based on client side detection methods. Having to wait for congestion to appear before attempting to reduce it is not a solution to the actual problem. The optimal way is to be able to avoid congestion in the first place. In order to achieve this, the next step must be reliable congestion feedback from the network itself. This feedback, in the form of ECN, would be impossible to misinterpret and could let the system react instantly instead of waiting for clear signs of congestion.

Only relying on inter-arrival delay and packet losses proved to result in a system that would have to send at a higher rate than what was achievable in order to detect congestion. Testing with competing video streams revealed that there is room for further improvement by for example monitoring the increase in measured round-trip time in order to detect when the network buffers are filling up slow enough to avoid detection by the inter-arrival time monitor. One could for example measure the rate of increase or have a set limit that defines the acceptable delay.

The key issue when using metrics at the client side is not only detecting congestion, but how to act based on the given information. As an example one can compare the behavior of the careful build with the aggressive build. Both detect congestion in the same way and react by lowering the bit rate in the same way. The difference lies in what the systems do after they have reached a state where they do not detect any congestion. The desired behavior is to be able to send at a high bit rate but with low delay, but that can be achieved in several different ways. This conclusion also shows that it was a good decision to focus on fewer metrics and spend more time on configuring the control systems instead of trying to add more ways of detecting congestion.

8.2 Goals and changes in the thesis

The study of previous solutions, requirements analysis and framework design processes proceeded as planned without any changes. The implementation shifted focus from several different components to getting the delay based adaptation working well. Things such as ECN and network based adaptation was looked at, but would require too much time to implement. Such additional features are left for future work. This thesis does identify the need for such components. The test cases were chosen to achieve equal or better performance than Google's implementation. Venturing further into changing and optimizing the implementation proposed in this thesis in cases where Google's underperform is also left for future work.

8.3 Requirements

The set of requirements proved useful when designing the framework and selecting what components that should be present. For the implementations, most of them were not explored fully due to time limitations. Many of them concern rate control strategies that require testing and adjustment both in controlled and live environments. During this thesis, focus was put on achieving good behavior in the first two test cases (static and temporary bottlenecks). Further test cases and requirements related to modules not implemented are left for future work.

1. Must prioritize low delay over high quality in the event of congestion
    a. This was not as straight forward as first thought as the case of TCP competition indicated that this may not always be true.
2. Must react based on information from several sources, including delay, packet loss and lower level feedback such as ECN.
    a. The main source of input during the testing was inter-frame delay. The results showed that there are cases where additional input is required to let the system identify what situation it is in.
3. Must be quick to react to network changes and persistent congestion

    a. After congestion is detected, the test implementations and Google's algorithm all reacted quickly. Maintaining acceptable bit rates during ongoing bottlenecks was also not much of a problem. This requirement was not too difficult to meet as long as the system is able to detect the congestion.

4. Should be able to differentiate between packet losses caused by congestion and other losses that may be unrelated to congestion.

    a. This was not explored and is left for future work.

5. Should keep a record of the rate history of specific connections

    a. This was briefly tested by setting a number of different starting rates. No plots are included from these tests. The result was that it reduced startup time as long as the available bandwidth was unchanged. This could be useful in cases where conditions rarely change, such as communication within local networks. In cases with mobile clients that rarely stay in the same network for long, this might be less useful. A reasonable guess could be made based on what type of connection the client is currently on (e.g. 3G, LTE, wired).

6. Must work together with different kinds of other traffic (web surfing, YouTube).

    a. Comparisons with TCP traffic was performed, but YouTube competition is left for future work. This is important information, but the number of different test cases would be very large.

7. Must work in different kinds of networks (wired networks, 3G, LTE and Wi-Fi).

    a. This is mainly left for future work.

8. Should take user preference on quality/delay into account.

    a. This is also left for future work. The practical implications of meeting this requirement in the testing would be to sometimes be satisfied with a lower maximum bit rate than what might be possible.

9. Should not generate more feedback messages than necessary.

    a. The feedback frequency used in all cases was the one originally used in Chromium. It in turn, was following the suggestions of the RTP related RFC:s.

10. Must utilize listen to feedback messages and be able to cope with missing ones, indicating congestion in the feedback channel.

    a. This was implemented, but not fully tested. It is left for future work where one might perform tests with impairments applied in both directions of the link.

11. Should include verification of the sender of the feedback messages.

    a. This is left for future work. While it is an important consideration, it would not have much impact on rate adaptation in an environment without malicious parties that try to sabotage the video session.

12. Should support input from other rate adaptation algorithms.

    a. This was discussed during the framework design phase, but never considered for implementation.

8.4 Adaptation behavior

8.4.1 Startup

As can be seen in the plots, the implementation proposed in this thesis as well as Google's uses a slow start approach. This is the simplest and safest approach, but not an optimal one. It may be the best approach when starting a video session in a network with unknown capacity. In such a case, starting slow and increasing until reaching the maximum rate or facing congestion is a good way of preserving low initial delay while finding the current network capacity. It is very important for the user experience to prevent high delay and choppy video during the initial "hello phase" of the conversation. Poor initial performance may instead turn it into a "can you hear me phase".

My design suggested that it could be useful to keep track of the obtainable bit rates during video sessions. Such data could be used to set a higher starting bit rate in video sessions between two clients who often communicate and have high available bit rates.

This scenario can be very likely within corporate networks where there might be a certain level of expected QoE. In other cases, a previously available bandwidth may not be available the next time the same users wish to open a video session. It is still very visible in the plots that there is a large amount of unused capacity available.

8.4.2 Reaction to congestion

8.4.2.1 Initial reaction to heavy congestion

The system can detect congestion in three ways; increasing inter-arrival delay, difference between incoming rate and target rate, and packet loss. In the case of congestion occurring due to a sudden reduction of available bandwidth, the overuse detector will react first.

In my earlier builds the system would heavily reduce the bit rate until the overuse detector declared that the inter-arrival time was no longer increasing. The system would then hold for a while before increasing again. The heavy rate reduction helped clear out the network queues quickly, but lead to poor network utilization. On the other hand, a small steady reduction would give higher network utilization, but at the cost of longer delay as the queues took longer time to empty. A viable solution turned out to be reducing the bit rate to 90 % of the estimated incoming bit rate as it allowed the queues to empty quickly while still not reducing the bit rate too far.

8.4.2.2 Careful increase

The careful state was introduced after it became apparent that the initial increase strategy was a bit too greedy when operating in a network with a lower available bandwidth than the maximum. The procedure of reacting to a “failed” increase attempt by going down to the previous rate often lead to the system remaining at a rate slightly above the available capacity. The careful approach was more passive and reacted to failed attempts by decreasing down to a previously established “safe” bit rate. This safe rate is increased when the sending rate has increased enough above it. Since the probing state was more careful than the standard increase state, finding appropriate thresholds and limits became more important as a failed increase attempt made the system back down and wait for a while before trying to increase again. This strategy is shown to be viable and provides a stable bit rate that is within the available capacity.

8.5 The need for network feedback

The test results highlight the need for several of the components present in the suggested design that were not implemented. This is especially true in the case of assistance from the network. In the simplest case, having an ECN capable network and a module that can detect ECN markings would let the network devices notify the clients that they should go down in bit rate even before the buffers had filled up and before the clients would have noticed a change in delay. Packets with ECN markings would be a clear signal that the sending rate must be adjusted immediately. A slowly increasing network queue could be undetectable by clients monitoring changes in RTT at the endpoints. An extreme case was shown in section 6.5. Changes in RTT could in some cases also be caused by heavy load at one of the clients.

Going into more advanced territory, having devices in the network that can perform assisted adaptation will be useful as well. In a case with several video streams competing over the same bottleneck, fairness would be assured if the clients can accept rate adaptation feedback from devices in the network that have the complete picture.

8.6 Impact on society and ethical aspects

The topic of rate adaptation is important as the use of bandwidth-heavy technology in our society is increasing. Effective adaptation could help reduce the stress put on our network infrastructure and utilize it more efficiently. This could in some cases reduce energy usage and the need for new infrastructure.

Working with content delivery also concerns the topic of Net neutrality. The goal of adaptation systems is hopefully always to give everyone the best possible user experience given the circumstances. It is however very possible to use the same techniques to give higher priority to certain types of content, or to certain content providers. Some systems may even be designed to disrupt network stability on purpose, as can be seen today with DDoS attacks. It must be ensured that development and standardization of Internet protocols and related work remains open and monitored by independent organizations.

8.7 Future work

8.7.1 Standardization of congestion detection and adaptation strategies

While it is clear that all video communication implementations participating in the same video session must use the same adaptation strategies to work properly, the same reasoning could be used on a larger scale.

In order to avoid network congestion and give all participants a fair share of the available bandwidth, all network protocols should use similar adaptation techniques and definitions of congestion. Otherwise, some participants might react to congestion while others do not. They might even detect that others are backing off and increase their bandwidth usage. Possibilities of a protocol-independent standard should be investigated.

8.7.2 More test cases

There are many more relevant test cases that may reveal other strengths and weaknesses of the systems. The ones presented in this thesis could be complemented by tests with more kinds of competing traffic as well as tests in live network environments. Live testing has the advantage of giving an idea of how the system performs in reality, but has the disadvantage of making it harder to understand certain behavior as there are more factors involved. An important case is one with congestion in both up and down links to see how the system can handle missing or late RTCP packets.

8.7.3 Implementation of ECN detection and other modules

While it can be argued that ECN markings and network based adaptation feedback will provide an improvement to performance, an implementation that use these features would be needed to prove it. Implementing support for them may be difficult as there does not seem to be any other implemented solutions available. Accessing the ECN bit in the IP header in application level systems may also be difficult. In general, implementing the entire proposed framework may not be possible on the Chromium platform. This should be considered the most important area to test in future research.

8.7.4 User-set priorities

The possibilities of user-set priorities detailed in the design should be explored further. This should be very helpful in the case with multiple competing video streams over a limited bandwidth. A case where such possibilities would be very helpful is The Tangibles student project at Luleå University of Technology [48]. The project consisted of creating a web based collaboration tool using WebRTC. Each user would have several video streams, one showing a shared sketching table and several others showing other participants. If the available bandwidth is limited, the sketching view would benefit the most of receiving a larger share of the bandwidth. Two possible approaches are being able to change the specific maximum bit rate for a stream or set a specific percentage of the available bandwidth that should be used. The first case is simpler as it can be done without having knowledge of any other parallel video streams.

8.8 Further reading

Several companies and groups are actively performing research in this area. The IETF has a working group called RMCAT [40], which is involved in standardization and research of congestion detection and rate adaptation. The documents and mailing lists there are a good place to start when looking for more information on this topic.

9 References

1 Floyd, S et al. *“TCP Friendly Rate Control (TFRC): Protocol Specification”,* RFC 5348, September 2008

2 Singh, Varun, Jörg Ott, and Igor DD Curcio. *"Rate adaptation for conversational 3G video."* INFOCOM Workshops 2009, IEEE. IEEE, 2009.

3 Barzuza, Tamar, et al*. "Trend: A dynamic bandwidth estimation and adaptation algorithm for real-time video calling."* Packet Video Workshop (PV), 2010 18th International. IEEE, 2010.

4 Rejaie, Reza, Mark Handley, and Deborah Estrin. *"RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet."* INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 3. IEEE, 1999.

5 Mel, Bob, Mats Björkman, and Per Gunningberg. *"Regression-based available bandwidth measurements."* In International Symposium on Performance Evaluation of Computer and Telecommunications Systems. 2002.

6 Ribeiro, Vinay, et al. *"pathchirp: Efficient available bandwidth estimation for network paths."* Passive and active measurement workshop. Vol. 4. 2003.

7 Keshav, Srinivasan. *"Packet-pair flow control."* IEEE/ACM Transactions on Networking (1995).

8 Wong, Starsky HY, et al. *"Robust rate adaptation for 802. 11 wireless networks."* International Conference on Mobile Computing and Networking: Proceedings of the 12 th annual international conference on Mobile computing and networking. Vol. 23. No. 29. 2006.

9 Chen, Xi, Prateek Gangwal, and Daji Qiao. *"RAM: Rate Adaptation in Mobile Environments."* Mobile Computing, IEEE Transactions on 11.3 (2012): 464-477.

10 Munaretto, Daniele, Dan Jurca, and Joerg Widmer. *"A Fast Rate-Adaptation Algorithm for Robust Wireless Scalable Streaming Applications."* Wireless and Mobile Computing, Networking and Communications, 2009. WIMOB 2009. IEEE International Conference on. IEEE, 2009.

11 Alay, Özgü, et al*. "Dynamic rate and FEC adaptation for video multicast in multi-rate wireless networks."* Mobile Networks and Applications 15.3 (2010): 425-434.

12 Ellis, Martin, Dimitrios P. Pezaros, and Colin Perkins. *"Performance analysis of AL-FEC for RTP-based streaming video traffic to residential users."* Packet Video Workshop (PV), 2012 19th International. IEEE, 2012.

13 http://www.netlab.tkk.fi/~varun/2013_MMSYS_Singh_MPRTP.pdf

14 Singh, Varun, Jörg Ott, and Igor DD Curcio. *"Rate adaptation for conversational 3G video."* INFOCOM Workshops 2009, IEEE. IEEE, 2009

15 Schulzrinne, H. Et al, *"RTP: A Transport Protocol for Real-Time Applications",* STD 64, RFC 3550, July 2003.

16 Wenger, S., et al*, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)"*, RFC 5104, February 2008.

17 http://tools.ietf.org/html/draft-alvestrand-rtcweb-congestion-03

18 http://www.tschofenig.priv.at/cc-workshop/irtf_iab-ccirtcpaper1.pdf

19 http://www.tschofenig.priv.at/cc-workshop/irtf_iab-ccirtcpaper1.pdf

20 Shalunov, S., and M. Kuehlewind. *"Low Extra Delay Background Transport (LEDBAT)"* (2011).

21 Kim, Jae-Gon, et al. *"An Optimal Framework of Video Adaptation and Its."* ETRI journal 27.4 (2005).

22 Semsarzadeh, Mehdi, and Mahmoud Reza Hashemi. *"A joint multi rate optimization framework for video adaptation in H. 264/AVC."* Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on. IEEE, 2010.

23 Westerlund, M., et al, *"Explicit Congestion Notification (ECN) for RTP over UDP",* RFC 6679, August 2012.

24 Floyd, S., and Jacobson, V., *"Random Early Detection gateways for Congestion."* IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.

25 http://queue.acm.org/detail.cfm?id=2209336

26 Zhu, X., Pan. R., *"NADA: A Unified Congestion Control Scheme for Real-Time Media"*, IETF Internet draft, work in progress, Sep. 2013.

27 Welzl, M., et al., *"Coupled congestion control for RTP media,"* IETF Internet draft, work in progress, October. 2013.

28 Handley, M., et al, *"TCP Friendly Rate Control (TFRC): Protocol Specification",* RFC 3448, January 2003.

29 Ramakrishnan, K., et al, *"The Addition of Explicit Congestion Notification (ECN) to IP"*, RFC 3168, September 2001.

30 Zhu, X., Pan. R., *"NADA: A Unified Congestion Control Scheme for Real-Time Media"*, IETF Internet draft, work in progress, Sep. 2013.

31 Landström, S, Larzon, L-Å, *"Reducing the TCP acknowledgment frequency."* *ACM SIGCOMM Computer Communication Review* 37.3 (2007): 5-16.

32 Carlberg, K. et al., *"Reactions to Signaling from ECN Support for RTP/RTCP,"* IETF Internet draft, work in progress, July. 2012.

33 Westerlund, M., et al, *"Explicit Congestion Notification (ECN) for RTP over UDP",* RFC 6679, August 2012.

34 Eardley, P., Ed., *"Pre-Congestion Notification (PCN) Architecture",* RFC 5559, June 2009.

35 Schulzrinne, H., et al, *"RTP: A Transport Protocol for Real-Time Applications",* STD 64, RFC 3550, July 2003.

36 Alvestrand, H. et al., *"RTCP message for Receiver Estimated Maximum Bitrate,"* IETF Internet draft, work in progress, January. 2013.

37 http://www.zti-telecom.com/EN/NetDisturb.html

38 http://apprtcsummertest.appspot.com/

39 http://apprtc.appspot.com/

40 http://datatracker.ietf.org/wg/rmcat/

41 http://www.html5rocks.com/

42 http://www.webrtc.org/

43 http://datatracker.ietf.org/wg/rtcweb/

44 Postel, J., *"User Datagram Protocol",* STD 6, RFC 768, August 1980.

45 Wenger, S., et al, *"Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)"*, RFC 5104, February 2008.

46 Sarker, ANM S. (2007). *A Study on Adaptive Real Time Video over LTE*. MSc Thesis. Luleå University of Technology.

47 Ott, J., et al, *"Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)",* RFC 4585, July 2006.

48 http://the-tangibles.blogspot.com/

49 http://c3lab.poliba.it/images/0/07/Webrtc_cc-Fhcmn2013.pdf

50 Perkins, C. S., Singh, V., *"Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions,"* IETF Internet draft, work in progress, July. 2013.