TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Juha Vihervaara

Dual-Mode Congestion Control Mechanism for Video Services

Julkaisu 1541 • Publication 1541

Tampere 2018

Tampereen teknillinen yliopisto. Julkaisu 1541
Tampere University of Technology. Publication 1541

Juha Vihervaara

Dual-Mode Congestion Control Mechanism for Video Services

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Auditorium 125, at Tampere University of Technology - Pori, on the 11th of May 2018, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2018

Doctoral candidate:	Juha Vihervaara Laboratory of Electronics and Communications Engineering Faculty of Computing and Electrical Engineering Tampere University of Technology Finland
Supervisor:	Tarmo Lipping, professor Laboratory of Signal Processing Faculty of Computing and Electrical Engineering Tampere University of Technology Finland
Pre-examiners:	Marilia Curado, assistant professor Department of Informatics Engineering University of Coimbra Portugal
	Pasi Sarolahti, university lecturer Department of Communications and Networking Aalto University Finland
Opponent:	Ismo Hakala, professor Information Technology Kokkola University Consortium Chydenius Finland

ISBN 978-952-15-4124-7 (printed)
ISBN 978-952-15-4128-5 (PDF)
ISSN 1459-2045

Juha Vihervaara, 2018. "Dual-Mode Congestion Control Mechanism for Video Services". Tampere University of Technology, Tampere, Finland.

Keywords: Congestion control, Video service

Abstract

Recent studies have shown that video services represent over half of Internet traffic, with a growing trend. Therefore, video traffic plays a major role in network congestion. Currently on the Internet, congestion control is mainly implemented through overprovisioning and TCP congestion control. Although some video services use TCP to implement their transport services in a manner that actually works, TCP is not an ideal protocol for use by all video applications. For example, UDP is often considered to be more suitable for use by real-time video applications. Unfortunately, UDP does not implement congestion control. Therefore, these UDP-based video services operate without any kind of congestion control support unless congestion control is implemented on the application layer. There are also arguments against massive overprovisioning. Due to these factors, there is still a need to equip video services with proper congestion control.

Most of the congestion control mechanisms developed for the use of video services can only offer either low priority or TCP-friendly real-time services. There is no single congestion control mechanism currently that is suitable and can be widely used for all kinds of video services. This thesis provides a study in which a new dual-mode congestion control mechanism is proposed. This mechanism can offer congestion control services for both service types. The mechanism includes two modes, a backward-loading mode and a real-time mode. The backward-loading mode works like a low-priority service where the bandwidth is given away to other connections once the load level of a network is high enough. In contrast, the real-time mode always demands its fair share of the bandwidth.

The behavior of the new mechanism and its friendliness toward itself, and the TCP protocol, have been investigated by means of simulations and real network tests. It was found that this kind of congestion control approach could be suitable for video services. The new mechanism worked acceptably. In particular, the mechanism behaved toward itself in a very friendly way in most cases. The averaged TCP fairness was at a good level. In the worst cases, the faster connections received about 1.6 times as much bandwidth as the slower connections.

Preface

This thesis contains the results of research work, which has been carried out at the Pori Campus of the Tampere University of Technology. This work has been a challenging journey, including both uphill and downhill phases. Fortunately, I was not alone on this road but accompanied by a group of people always willing to coach, support, help, and motivate me. Certainly, I would have never reached the endpoint of this trip without the help and support of others. For this reason, I would like to thank all the people who have been involved in the process of creating this thesis throughout the years.

First, I would like to thank my supervisors, Professors Pekka Loula and Tarmo Lipping, for taking me on as their student and guiding me through this study. They have been the ideal advisors in every respect. Secondly, I would like to thank my colleagues and fellow students for their support. I would also like to thank the pre-examiners Marilia Curado and Pasi Sarolahti for their constructive comments and criticism, which have been invaluable for the completion of this thesis.

This work has been supported financially by the Ulla Tuominen Foundation, the High Technology Foundation of Satakunta, and the Finnish Cultural Foundation. I wish to express my thanks to these foundations.

I owe a special debt of gratitude to my parents and family. They, more than anyone else, have been the reason I have been able to get this far.

Table of Contents

Abbreviations

ACK	Acknowledgement
ADSL	Asymmetric Digital Subscriber Line
AIAD	Additive Increase, Additive Decrease
AIMD	Additive Increase, Multiplicative Decrease
API	Application programming interface
AQM	Active Queue Management
ARQ	Automatic Repeat Request
CCID	Congestion Control Identity
CDN	Content Distribution Network/Content Delivery Network
CE	Congestion Experienced
CoDel	Controlled Delay
CVIHIS	Congestion Control for Video to Home Internet Service
CVIHIS-BLM	CVIHIS Backward-loading mode
CVIHIS-RTM	CVIHIS Real-Time Mode
CWR	Congestion Window Reduced
DASH	Dynamic Adaptive Streaming over HTTP
DCCP	Datagram Congestion Control Protocol
DNS	Domain Name System
DupACK	Duplicate Acknowledgement
DVD	Digital Video Disc
ECC	Explicit Congestion Control
ECN	Explicit Congestion Notification
ECT	ECN Capable Transport
ETFRC	Enhanced TCP-Friendly Rate Control
EWMA	Exponentially Weighted Moving Average
FEC	Forward Error Correction
FTP	File Transfer Protocol
GCC	Google Congestion Control for Real-Time Communication on the World Wide Web
HAS	HTTP Adaptive Streaming
HTTP	HyperText Transfer Protocol
ICC	Implicit Congestion Control
IETF	Internet Engineering Task Force
ISP	Internet Service Provider
LDA+	Enhanced Loss-Delay Based Adaptation Algorithm
LEDBAT	Low Extra Delay Background Transport
MD	Multiplicative Decrease parameter of CVIHIS
MIAD	Multiplicative Increase, Additive Decrease

MIMD	Multiplicative Increase, Multiplicative Decrease
MPD	Media Presentation Description
MSS	Maximum Segment Size
NS-2	Network Simulator Version 2
Nam	Network Animator
PCN	Pre-Congestion Notification
PF	Pushing Factor parameter of CVIHIS
QoE	Quality of Experience
QoS	Quality of Service
RAP	Rate Adaptation Protocol
RED	Random Early Detection
RFC	Request For Comments
RSVP	Resource Reservation Protocol
RTCP	RTP Control Protocol
RTO	Retransmission timeout
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
RTTVAR	Round-Trip Time Variation
SACK	Selective Acknowledgement
SCTP	Stream Control Transmission Protocol
SF	Smoothing Factor parameter of CVIHIS
SRTT	Smoothed Round-Trip Time
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TCP-LP	TCP Low Priority
TEAR	TCP Emulation at Receivers
TFRC	TCP Friendly Rate Control
TFWC	TCP Friendly Window-based Congestion Control
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VBR	Variable Bit Rate
VoD	Video-on-Demand

1 Introduction

1.1. Background

The Internet is a global system for interconnecting communication networks. It is a network that consists of thousands of different kinds of networks. There are small and big networks ranging from local scope to global scope. There are private, public, academic, business, and government networks. These networks use different kinds of technologies at the bottom of protocol stacks, for example, wireless and optical networking technologies. However, there is one factor common to all of these networks. They all use the TCP/IP (Transmission Control Protocol/Internet Protocol) suite (Kurose and Ross 2017; Stallings 2013; Tanenbaum and Wetherall 2010) for communication in the upper parts of the protocol stack. The Internet uses the standardized Internet protocol suite to serve billions of users worldwide. The Internet and TCP/IP have been developed together so that TCP/IP offers the mechanisms for implementing the Internet. Since the Internet has been undergoing change throughout its lifetime, TCP/IP has also continued its evolution to meet the needs of the Internet. This development of the Internet, and therefore the development of TCP/IP, will continue so that the new requirements of Internet users can be realized. Internet protocols are specified in open standards, known as RFCs (Request for Comments).

IP is a network layer protocol of the TCP/IP protocol suite. IP is responsible for delivering data packets to the right destinations. It is a connectionless datagram based protocol. It is a simple protocol that does its job with the help of other protocols like routing protocols. Because IP is a simple protocol, the transport services needed by applications must be offered on top of the IP protocol. For this reason, the services required must be implemented as part of the transport layer or by the application itself. TCP and UDP (User Datagram Protocol) are both built on top of IP. These protocols are the core protocols of the transport layer. TCP and UDP operate in a very different way, and the one which is used depends on the requirements of the application process.

TCP is a byte-oriented protocol that guarantees the delivery of data bytes. This protocol offers error control mechanisms for the use of applications. TCP uses acknowledgments and retransmissions to implement reliable data delivery. Duplicate packets are discarded and out-of-order packets are resequenced by TCP. Therefore, data packets are delivered to the application in the order in which they were sent. TCP is a reliable and connection oriented protocol. A logical connection must be established between the end points before data transmission. Thus, the implementation of TCP is heavyweight. On the other hand, if the application requires the guaranteed delivery of data, TCP is the right choice for the transport layer protocol.

However, TCP is not an ideal protocol for the use of all video services (Choi et al. 2012). Some real-time applications do not want to use the retransmissions offered by a transport protocol because such kinds of applications are often loss-tolerant and played in real time. These applications are loss-tolerant because some packet drops do not significantly degrade the quality of service experienced by the users of these applications. These packet drops can be softened by using the error correction properties of the applications. Because they work in a real-time repeat mode, these kinds of applications do not always have enough time for retransmissions. The received packet must be presented on the screen of a user device just after it has entered the destination device. In the case of TCP, out-of-order packets also cause a problem, known as head-of-the-line blocking. Due to the order delivery property of TCP, the bytes after missing ones cannot be delivered to the application. If a more recent packet arrives, this new data must be put in a queue. The application cannot access this new data until the lost bytes are received. TCP's burst-like transmission also causes delay jitters and sudden quality degradations because there can be abrupt and deep sending rate reductions.

In contrast, UDP is often considered more suitable for real-time video applications. It offers two services for applications. First, it provides a way to distinguish between the multiple applications running on a single host. Like TCP, UDP uses so-called port numbers for implementing this service. Second, UDP also offers optional error checking for applications. Due to these minimal services, UDP is a light protocol on top of IP. It is a connectionless unreliable message-based protocol. Message-based means that it preserves message boundaries. However, it is possible that nothing will be received if the message is dropped due to congestion or error checking because UDP is an unreliable protocol without packet retransmissions. It is up to the application to split data into messages and also to provide all the error recovery functions for dropped packets that are required. Because of this simplicity, UDP makes it possible to send streaming media over the networks effectively. It provides a way to meet the real-time demands of applications as closely as possible. Therefore, UDP has typically been used by applications such as real-time media (audio, video, Voice over IP) where the on-time arrival of packets has been more important than reliability. Simple query/response applications also use UDP because there is no overhead for setting up a reliable connection.

Unfortunately, even UDP is not an optimal protocol for all real-time media applications. It cannot offer all the services needed by some applications. Therefore, an extra-protocol called Real-time Transport Protocol (RTP) (Schulzrinne et al. 2003) has been added on top of UDP. There is, however, one major difference between UDP and RTP. UDP is part of the kernel whereas RTP is part of the application process. With this choice, RTP was implemented without the need for modifying the socket API. In that way, it is also possible to modify RTP for the needs of an application. This enables developers to implement congestion control as part of the RTP protocol. A complete implementation of

RTP for a particular application type requires a separate profile and payload format specification. For example, Wang et al. (2011) define a profile for RTP with H.264 video. RTP can also offer special services to applications because there is an optional extension header part at the end of the RTP header.

As stated above, real-time multimedia applications prefer the timely delivery of information, and these applications can often tolerate some packet losses to achieve this goal. RTP provides facilities for loss detection and out-of-sequence arrival detection that are common during transmission over UDP/IP. RTP also offers jitter compensation. Multimedia data messages are framed and transmitted using RTP packets, equipped with appropriate timestamps, and increasing sequence numbers. Depending on the RTP profile in use, the *Payload Type* field is set to the appropriate value. The RTP receiver receives these RTP packets, detects missing and reordered packets, and may perform jitter compensation. The frames are decoded based on the payload format and then presented to the end user.

RTP can be used in conjunction with the RTP Control Protocol (RTCP) (Schulzrinne et al. 2003), although the presence of RTCP is optional. While RTP carries media streams, RTCP is used to monitor the quality of data streams and collect statistics information such as transmitted octet and packet counts, lost packet counts, jitters, and round-trip times. Applications can use this information to control the values of sending parameters. For example, if many packets are lost, the sending rate can be reduced by using a different codec. RTCP is implemented using an out-of-band signaling principle. This means that RTP and RTCP use different port numbers. Typically, an RTP data stream is using an even-numbered port, and RTCP control messages for that data stream are sent over the next higher odd-numbered port. The bandwidth of RTCP traffic compared to RTP traffic is small, typically around 5% of the latter. Therefore, there is no danger of reverse path congestions.

There is one more significant difference between TCP and UDP protocols, which is the most important regarding this thesis, i.e., congestion control. TCP implements congestion control whereas UDP does not. In fact, no congestion control was available on the Internet in its early years. However, in the late 1980s, the Internet experienced a number of congestion collapses. In response to this, Jacobson proposed a series of mechanisms which the sender could implement, known collectively as “Congestion Avoidance” for TCP (Jacobson 1988). The main modification was the introduction of a new variable to control the sending rate. This parameter was called a congestion window. The novel congestion control implementation introduced by Jacobson was clever, because it required changes only to the sender side. Due to this approach, it could be deployed gradually. In contrast, UDP continued without congestion control. In those days, this was reasonable because most of the traffic was TCP-based, and UDP was only used with short-lived request/response connections.

By the end of the 1990s, it could be seen that UDP-based long-lived connections would become more common. Floyd and Fall (1999) pointed out that, without the appropriate congestion control mechanisms of UDP flows, TCP connections would receive a much smaller bandwidth share than competing UDP flows. Therefore, it was no big surprise that interest towards UDP-based congestion control increased and a lot of research work was done (Widmer et al. 2001). Some proposals were built on top of UDP and some on top of RTP. The main effort was the development of the Datagram Congestion Control Protocol (DCCP) (Kohler et al. 2006b). Instead of using UDP as part of congestion control based communication, the developers of DCCP decided to design a totally new protocol. This new protocol was placed as part of the transport layer and, thus, as part of the kernel. DCCP does not include any kind of fixed congestion control mechanism. It is possible to use DCCP with different kinds of congestion control mechanisms, each suitable for a certain application group.

Despite great research work, the congestion control mechanisms developed for UDP-type traffic are not widely used by network operators. There has not been any actual need for the use of these congestion control solutions. Dynamic rate adaptation techniques have been developed so that TCP-based video streaming can be implemented in a reasonable way in existing overprovisioned networks. Network operators have relied on overprovisioning in order to avoid congestion in their networks. Overprovisioning means that a network operator purposely offers an unnecessary amount of network capacity to support peak demand without significant service degradation. Because overprovisioning is commonly used to protect networks against massive traffic variations, it can be said that problems have been solved by increasing the bandwidth of network links and the processing power of routers. In particular, overprovisioning of network links has become common because optical fiber offers a very economical way to perform overprovisioning.

Overprovisioned networks are considered to be cost-effective to manage and easy to troubleshoot. Typically, such networks will not cause problems unless there is a massive burst of traffic or network link failure. Therefore, it comes as no surprise that there was no notable criticism of overprovisioning for many years. However, nowadays there are some arguments against *massive* overprovisioning. For example, massive overprovisioning with unnecessary high power consumption goes against green Internet ideology (Gupta and Singh 2003; Bianzino et al. 2012). Therefore, alternative approaches are welcome, and one such approach is to improve the congestion control mechanisms of the Internet.

Developing techniques for Internet video streaming is of major importance today because various Internet-based video services have found their way into common use. The reason for this is that a few years ago network operators started to offer high-speed Internet connections at affordable prices (Choi et al. 2012) making it possible to offer films, sport events, music concerts, and TV programs to home customers via the Internet. These programs can be watched either real-time or non-real-time. Some programs are offered in high-definition quality. In Finland, for example, many such services are available, offered by network operators and media houses, for example, Elisa Viihde, MTV3 Katsomo, Sonera Viihde, and Netflix.

In the near future, the volume of video services will continue to increase if Cisco's forecast is correct. Some figures for the recent past and the future forecast are shown below.

Cisco’s forecast (Cisco 2016a) presents the following video highlights:

- Internet video will increase four-fold between 2015 and 2020.
- Every second, nearly a million minutes of video content will cross the network by 2020.
- Globally, IP video traffic will be 82 percent of all consumer Internet traffic by 2020, up from 70 percent in 2015.
- Internet video to TV grew 50 percent in 2015. Internet video to TV will continue to grow at a rapid pace, increasing 3.6-fold by 2020. Internet video-to-TV traffic will be 26 percent of consumer Internet video traffic by 2020, up from 24 percent in 2015.
- Consumer Video-on-Demand traffic will nearly double by 2020. Ultra-high definition (UHD) will be 20.7 percent of IP Video-on-Demand traffic in 2020, up from 1.6 percent in 2015.
- Globally, virtual–reality traffic will increase 61-fold between 2015 and 2020.

Cisco’s forecast (Cisco 2016b) presents the following mobile video highlights:

- Three-fourths of the world’s mobile data traffic will be video by 2020.
- Mobile video will increase 11-fold between 2015 and 2020.

As can be seen, in the future there will be more video traffic on the Internet. Although some video services can use TCP to implement their transport services in a manner that actually works, the growth of video traffic means that there will always be a need for better transport services and congestion control mechanisms. This will give a new opportunity for deployment of both old and new congestion control mechanisms.

1.2. Problem statement and the aims of the thesis

As explained earlier, video-based services have become a common source of network traffic, and this growing trend is set to continue in the near future. Broadly speaking, based on these changes in Internet traffic types, there are two possibilities for implementing congestion control. First, significant overprovisioning can be used to ensure the proper functioning of networks. The second possibility is to use only slight overprovisioning and equip all video traffic with appropriate congestion control. The current solution, where congestion control is based on the TCP protocol, is not necessarily the most optimal solution for two reasons. The first reason is that TCP's current congestion control mechanisms are not ideal for the use of all video services. The second reason is that traffic flows that do not implement congestion control could harm TCP.

There are different kinds of ways to use video over the Internet. With real-time streaming, only a moderate buffering can be used on the receiver side due to the real time demands. Therefore, delay and bandwidth demands are important. On the other hand, if a video can be downloaded to the user device before it is watched, the delay and bandwidth demands are not important, and some kind of background loading can be used. There may also be some kind of intermediate form in which the event is recorded on the server owned by the service provider. This recorded video will be watched later in a real-time manner. In this case, buffering can be utilized on the receiver side. At first, the video can be transferred at high speed. When there is enough data in the receive buffer, the transfer mode can be switched to the background loading type. Perhaps with an appropriate pricing policy, the service provider could favor background loading type services. In addition, cache servers can be used so that videos can be stored near the customers. In this case, the service provider can load films, for example, to cache servers by using background loading.

Based on the previous paragraph, we can see that there are two different kinds of transfer modes needed for video services. There is a need for a background-loading mode. When this mode is used, the video download can be performed in a tardy manner if the network load is high. Henceforth, we refer to this mode as the backward-loading mode. With this mode, delay and bandwidth demands are moderate. On the other hand, there is also a need for a real-time mode where delay and bandwidth demands are important. Based on these different kinds of demands, these two modes also need different kinds of congestion control mechanisms. The backward-loading mode has to work like a low-priority service in which the bandwidth is given away to other connections when the load level of the network is high enough. In contrast, the real-time mode always demands its fair share of the bandwidth.

There are many congestion control mechanisms suitable for either low-priority service or real-time service. However, there has been little research work on developing an integrated mechanism suitable for both modes. The aim of this thesis is to design just such an integrated congestion control mechanism. In this case, the term integrated means that these two modes are as convergent as possible, although they cannot be totally identical due to the different kinds of demands. Of course, this kind of dual-mode mechanism could be realized by putting together the best low-priority and real-time algorithms. However, such a solution would be clumsy, especially in cases where mode changes have to be made "on the fly". The real dual-mode mechanism presented in this thesis makes it possible to switch between the modes in a seamless way.

The main objective of this study is to develop the dual-mode congestion control mechanism for video services. In addition, the aim of the study is that the features of the new mechanism are better suited to existing video streaming practices than the features of TCP's congestion control mechanisms.

In the following section, some targets will be set for this dual-mode mechanism. Typically, this kind of service is implemented in a client-server manner. This means that one video server serves a great number of clients. Therefore, the natural choice is that the algorithm of this mechanism is receiver-based. This means that the aim is to carry out all possible conclusion procedures on the receiver side. If we put this receiver side implementation principle together with the fact that there are differences between the algorithms of the modes, we obtain a new target. We can set this target so that there can be differences between the modes on the receiver side but the sender side should be as similar as possible for both modes. The real-time mode wishes to receive its fair share of network bandwidth. Because TCP has been the most widely used congestion control orientated protocol in recent years, this fairness comparison is traditionally made with the TCP protocol. This means that this mode should be TCP-friendly. As is usual with these kinds of video applications, this mode should use bandwidth in a much smoother way than TCP's congestion control does. Of course, this new mechanism should be stable and scalable, which are typical requirements for all congestion control mechanisms.

Instead of specifying the complete protocol, the aim of this thesis is to specify only a congestion control mechanism that could be used in different parts of a protocol stack, as well as cooperation with existing protocols. It could be used as part of a transport protocol such as DCCP, as part of an application, or as part of the RTP protocol. For this reason, packet formats, hand-shakes and reliability issues will only be discussed in brief.

1.3. Structure of the thesis

The structure of this thesis is presented next. In Chapter 2, the principles of congestion control are presented. For example, the AIMD principle behind sending rate adaptation is explained. The fairness issues of congestion control are also discussed. Nowadays congestion control is implemented partially by overprovisioning. Therefore, motivations related to overprovisioning are also described.

The congestion control and video transfer mechanisms of the Internet are explained in Chapter 3. In this chapter, Internet congestion control is defined loosely. The chapter introduces many mechanisms widely used on the Internet today. On the other hand, Chapter 3 also includes many mechanisms that are only at the level of proposals. Some of these proposals may be seen in use in a few years' time but some may never be deployed. In section 3.1, TCP congestion control mechanisms are presented. TCP congestion control is handled because new congestion control mechanisms are often designed to be TCP-friendly. Section 3.2 takes a look at delay-based congestion control since the mechanism proposed in this thesis utilizes delays for observing an incipient congestion. In this thesis, a tailored congestion control mechanism for video services will be developed. It is therefore natural that some congestion control proposals for video services are presented in section 3.3. Section 3.4 presents the video streaming principles of the Internet.

In Chapter 4, the dual-mode congestion control mechanism for video services is designed, simulated and tested in a real network. At first, some elementary choices are made and the argumentations behind these choices are explained. For example, the explanation is given why a delay-based congestion control approach has been used as part of the implementation. The behavior of the new mechanism and its friendliness with the TCP protocol have been mainly investigated by means of simulations. After comprehensive simulations, the real network version has been implemented and tested.

Only the basic functions are included in the developed congestion control mechanism in Chapter 4. In Chapter 5, some refining and elaboration work is presented. For example, there is an analysis of whether the developed mechanism is capable of multicast delivery after some modifications.

Chapter 6 summarizes the main contributions and results of this thesis. This chapter also introduces some proposals for future work.

2 Review of Congestion Control

The principles of congestion control are introduced in this chapter. Congestion control is an extensive research area and only the factors relevant to this thesis are presented here.

2.1. Background of congestion control

TCP/IP networks, such as the Internet, are especially open to congestion because of their basic connectionless nature on the network layer. IP routers do not support resource reservation in a normal situation. Instead, packets of variable sizes are sent by any host at any time. This makes it difficult to predict traffic patterns and resource needs. While connectionless networks have their advantages, robustness against congestion is not one of them.

It is not against the Internet's laws that packets are queued or dropped inside a network because, in its basic form, the Internet does not provide any guaranteed quality of service. Because a congestion situation is realized through the queuing and dropping of packets inside the network, congestion situations are also acceptable. From a network user's point of view, congestion is not a desirable situation. Hence, a network with user-friendly properties must implement some kind of congestion control.

If congestion control is not implemented, there is a threat of serious trouble. When some part of the network is in a congested state, it queues traffic and some packets are even dropped. Therefore, receivers do not receive expected packets on time, and senders cannot get acknowledgements inside the time limits. After that, senders that implement reliable communication will begin to resend packets, and these new packets will cause further congestion. Such a situation can lead to an extreme congestion situation called a congestion collapse. Congestion collapse (also called congestive collapse) is a situation when little or no useful communication takes place due to congestion. Congestion collapse was first explored as a possible problem at the beginning of the 1980s by Nagle (1984). There can be many reasons why networks become congested. The paper by Singh et al. (2008) lists the following reasons: limited memory space, limited channel bandwidth, limited router capacity, load of the network, link failure, heterogeneous bandwidths. A detailed discussion of these factors is given in their paper. Floyd and Fall (1999) also explain reasons behind congestion collapse.

There are many definitions for the term network congestion. Some examples are:

- congestion occurs when resource demands exceed the capacity of a network (Jain 1990);

- congestion occurs when too many packets are present in a network, and therefore, packets are lost or significantly delayed;
- a network is said to be congested from the perspective of a user if the service quality noticed by the user decreases due to the increase of the network load;
- when a network is congested, increasing the offered load only leads either to a small increase in the network throughput or to an actual reduction in the network throughput (Kurose and Ross 2017, p. 295).

The third definition emphasizes the fact that it is the network user who ultimately decides if the quality of the service is good enough. The fourth definition is the most technical. The offered load is the amount of data that is sent to the network, such as the number of packets per second. Throughput is the amount of data that passes through the network and is received on the receiver side. In a congested situation, the throughput cannot increase because the resources of the network are already fully utilized. If the offered load is increased by sending extra packets to the network, these extra packets must be dropped. The throughput can even decrease if packets are dropped at the beginning of the connection path, and therefore, the routers at the end part of the connection become underutilized (Floyd and Fall 1999).

The goal of congestion control is to avoid a congestion situation in network elements. There is also a more sophisticated definition for congestion control. This definition says that the target of congestion control is to adapt the sending rates of senders to match the available end-to-end network capacity. This definition emphasizes the fact that network-wide approaches must be used to implement congestion control. Otherwise, congestion is only moved from one node to another. In theory, traffic should be monitored over the whole network.

The background of congestion control lies in queuing theory. In a packet-switched network, packets move into and out of queues when these packets traverse through a network. As a result, packet-switched networks are often said to be networks of queues. These queues are also called buffers. It is typical for packet-switched networks that packets may arrive to a router in bursts. The task of buffering is to absorb data bursts inside the network. The presence of buffers is essential to permit the transmission of bursty traffic. Without buffers, a lot of packets would be discarded. However, it is also a target to empty buffers during silent periods. Queuing inside the network is not a desired operation. The queue limits should not reflect the steady lengths of queues that we want to maintain in the network; the lengths of queues should reflect the size of bursts we need to absorb (Braden et al. 1998).

There are two schemes for implementing congestion control. These schemes are the congestion avoidance scheme and the congestion control scheme (Jain 1990). These schemes are related to each other, but also distinct. They are related because both solve the problem of congestion management in the network. The congestion avoidance scheme allows networks to operate in a region where delays are low and throughput high. It tries to protect the network so that the network does not enter the congested state. In that way, the congestion avoidance scheme tries to avoid packet drops. In contrast, with congestion control schemes, packet drops are signals for the control function to react. After a packet drop, the congestion control scheme tries to bring the network back to an operating area with no drops. The congestion avoidance scheme is a preventive mechanism whereas the congestion control scheme is a recovering mechanism. It has also been said that congestion avoidance is a proactive approach and congestion control a reactive approach (Tian et al. 2005).

The difference between these two schemes is rather subtle. If the congestion avoidance scheme is used, the congestion control scheme is still required because congestion avoidance cannot be trusted to keep the network at its optimal area in all circumstances. Of course, if the network is working in the connection-oriented mode, the congestion avoidance scheme is sufficient alone, as the problem of congestion is solved by reserving resources along the connection path during the connection setup phase. Often concrete mechanisms include both these features. In this thesis, the term congestion control is used in a general sense for congestion control functions. The term congestion avoidance is used only if we want to emphasize the congestion avoidance feature.

The Internet works in the best-effort way. There are also aspects where the Internet should support quality of service (QoS) principles. A lot of research work has been done in this area (Xiao and Ni 1999; Meddeb 2010). A header field of the IP protocol has been reserved for this QoS function. However, the large-scale use of QoS has not been realized in practice. For this reason, for example, the loading times of web resources must be optimized to be fast enough by other methods (Vihervaara et al. 2016a). In this thesis, the QoS aspect is not considered. However, it is worth remembering that congestion control and QoS issues are bound together. For example, routers have to pay attention to which packets may be queued or dropped. If some connections are promised that they will experience low delays and low drop ratios, then other connections have to do congestion control work on behalf of these connections. The dual-mode mechanism developed in this thesis could be achieved using QoS techniques. In this case, the mechanism could be called the dual priority mechanism. However, this kind of mechanism could not be a pure end-to-end mechanism because the implementation would require network support at least to some extent.

It can be difficult to utilize network resources in an efficient way if there is no kind of congestion control in a network, or if the congestion control is implemented inappropriately. Costs related to bad congestion control are underutilized network resources and modest performance received by applications. The consequences of bad congestion control are described in Kurose and Ross (2017, pp. 289-295). The four cost principles of Kurose and Ross (2017) point out that overprovisioned networks can also derive advantage from proper congestion control. These principles are explained below.

The first cost of a congested network is that large queuing delays are experienced after the average packet arrival rate nears the link capacity. Matters are even worse if we take into consideration that the nature of network traffic is sometimes self-similar (self-similar traffic is explained in section 2.9). This first principle also points out why a theoretical option, using infinite buffer space in routers, is not a sensible solution against congestion losses. One important property for a good congestion control mechanism can be derived based on these queuing delays. A good mechanism tries to keep queues short.

The second cost of congestion is that the sender must perform retransmissions in order to compensate the packets that are dropped due to the buffer overflow. This automatically means that the offered load is bigger than the throughput. Of course, this is only possible if we are using a reliable protocol with retransmissions between the sender and receiver. An unreliable protocol without retransmissions can also be harmful in this regard. If an unreliable protocol works without congestion control, it can, with its high sending rate, fill the router buffers. In this way, it can force reliable protocols into packet losses and retransmissions. On the Internet, this means that UDP traffic sources can be harmful to the proper functioning of the TCP protocol. UDP sources can force TCP sources into retransmissions.

The duration between sending a certain packet and receiving the corresponding acknowledgement for that packet is called round-trip time (RTT), also known as round-trip delay. When a router is in a congested state, queuing delays increase. Because delays increase, round-trip times (RTT) also increase. These increased RTTs may lead to a situation in which retransmission timeout (RTO) timers expire and therefore some received data packets will be resent unnecessarily. This yields the third cost of congestion because after an unnecessary retransmission, routers have to use their resources to forward this unneeded copy of the packet. This also shows why proper RTT estimations are important as well as good RTO calculation rules based on RTT estimations (Jain 1990).

When a packet is dropped along the path, transmission capacity is wasted because the transmission capacity that was used at each of the links to forward that dropped packet is wasted. This is the fourth cost of congestion. This means again that the offered load is bigger than the throughput. This is depicted in Figure 2.1.

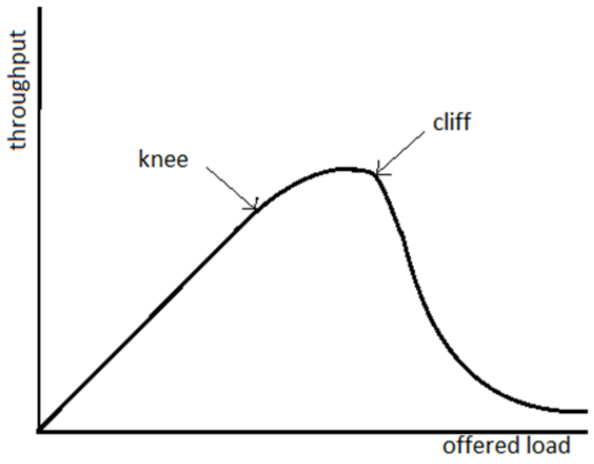


Figure 2.1 Congestion collapse

In Figure 2.1, the offered load is presented on the x-axis and the throughput on the y-axis. It is assumed that the traffic is bursty. Situations without the implementation of proper congestion control will eventually lead to a phenomenon called congestion collapse. Congestion collapse happens after a point called “the cliff”. After that point, the throughput decreases dramatically because the transmission capacity of the routers is wasted on retransmissions and packets dropped later along the path. In Figure 2.1, there is also a point called “the knee”. After this point, the throughput increases more slowly than the offered load. This means that some of the packets are dropped due to congestion. It can be said that the target of a congestion control mechanism is that the offered load level is close to the knee. The target is more an area than a point. It is difficult to define the exact location of the target because delays increase dramatically when the traffic rate nears the capacity of the routers.

The operation areas of congestion control and congestion avoidance can now be highlighted based on the paper by Chiu and Jain (1989). The congestion avoidance scheme operates below the knee point and tries to prevent packet drops. The congestion control scheme operates after the knee point, and hopefully far enough from the cliff. When the network load approaches the cliff, the control function typically takes a long step to the left so that the offered load is reduced to below the knee point.

2.2. Queue management

If the service rate of a router is less than the arrival rate of packets, then packets are queued into the buffer of that node. If the arrival rate remains over the capacity of the router for a long time, the buffer will eventually overflow. There are several reasons why queues of endless length are not desirable and why these kinds of endless queues cannot solve the problem of overflowing queues. The first reason is the self-similarity nature of Internet traffic, which is described in more detail in section 2.9. The second reason is the default behavior of the TCP protocol. Because TCP uses implicit congestion control, it relies on packet losses as a congestion indicator, and at the same time, also an indicator for the capacity limit of the network. The sending rate of TCP sources will increase until the length of the queue grows beyond its limit, no matter how long the queue is. The next reason is the growth of queuing delays as the length of queues grows. The final reason against long queues is the fluctuations of round-trip times. Long queues distort RTT samples and thus have a negative effect on any RTT-based mechanism. For instance, it is difficult to estimate when retransmissions should be triggered.

Avrachenkov et al. (2005) have analyzed the optimal choice of buffer sizes. The most common rule of thumb states that the buffer length should be set to the bandwidth delay product of the network. The bandwidth delay product is the product of the link capacity and round-trip time. In this case, the delay is the same as the average round-trip time of the connections traveling through the router. This rule comes from the simple presumption that TCP stops sending packets for the duration of one round-trip time after a single packet loss. If we wish to use the congested link in an efficient way during the congestion recovery phase, the buffer must have enough stored packets for sending for the duration of one round-trip time. Due to traffic aggregation, several research studies have suggested that the buffer size should be set to a smaller value. Appenzeller et al. (2004) suggest that it would be better to divide the bandwidth delay product by the square root of the number of flows traveling through the router. Avrachenkov et al. (2005) recommend that buffer sizes can be used even smaller than those suggested by Appenzeller et al. (2004).

Because the buffer size is always finite, packets may be dropped. Queue management algorithms manage the length of queues by dropping packets in an appropriate manner when necessary. There are three basic dropping techniques (Jain 1990). If the node drops packets only when its queue is full and the last packet to arrive is dropped (and so resides at the tail of the queue), the node is said to use a tail drop technique. In addition to the tail drop, two alternative dropping techniques can be applied when queues become full. These dropping techniques are the random drop and front drop. Under the random drop, the router drops a randomly selected packet from the queue. The random drop is a rather expensive operation since it requires moving back and forth in the memory space of the queue when packets are dropped. Using the front drop, the router drops the packet at the front of the queue when the queue is full and a new packet arrives. These three dropping techniques have a so-called full-queues problem, meaning that queues become full and overflow without any signal of rising congestion.

A solution to the full-queues problem is to drop packets before a queue becomes full so that end nodes can respond to congestion before the buffer overflows. Such proactive approaches are called active queue management (AQM) techniques. By dropping packets before the buffer overflows, AQM allows routers to select when and how many packets to drop. One of its main targets is to avoid the cluster type of packet drop events. In this way, AQM can improve the efficiency of the TCP protocol.

2.2.1. Traffic phase effects and active queue management

The problem of the traffic phase effect was introduced by Floyd and Jacobson (1991). They describe how phase differences between competing traffic streams can be the dominant sources for the relative throughput differences between these streams. Floyd and Jacobson (1991) show by means of simulations that drop tail gateways can cause systematic discrimination against some connections in TCP/IP networks with strongly periodic traffic. Often there is no clear reason for such discrimination. The discriminated connection simply has the bad luck to transmit packets at the wrong time instances when the queue is full. In addition, this discriminated connection does so in a periodic manner.

Some of the traffic on the Internet is highly periodic either because of the periodic character of the traffic (real-time speech, some video types) or because the window-based control protocol has a periodic cycle equal to the round-trip time of the connection. In window-based control, the sender is allowed to send a certain maximum number of packets to the network between congestion feedbacks. This number of packets is called the current window and, for a particular connection, the number of outstanding packets is controlled by this current window. Typically, the sender sends the allowed number of packets to the network as quickly as possible. When the receiver receives these packets, it immediately sends an acknowledgment (ACK) packet in response to the received packets. When the source receives this ACK, it immediately transmits another window of data packets. The round-trip time of the connection is the source of the periodic traffic in this case.

One special case of the phase effect is global synchronization. In global synchronization, many connections decrease their sending rate at the same time after congestion, because the router drops packets in clusters due to the high level overflow of the buffer. After this decrease event, the network is massively underutilized and the sources can now increase their sending rates. If the RTTs of the connections are equal enough, the same kind of overflow can be repeated in a synchronized manner. Nowadays, when networks are very heterogeneous, the problems with the traffic phase effect and global synchronization are more rare.

The phase effect and global synchronization problems can be eliminated by adding an appropriate randomization to the network (Floyd and Jacobson 1991). The randomization can be placed in routers or end-systems. However, because the negative impacts of phase effects only appear when packets are dropped by routers due to congestion, the natural place for randomization is the queue management mechanisms of routers. Analysis suggests that simply coding a router to drop a random packet from its queue in an overflow situation, rather than dropping it from the tail, is often sufficient. Because this kind of middle-drop is an inefficient way to drop packets, there is a need for more sophisticated AQM mechanisms to solve the phase effect problem.

A lot of research work has been done concerning AQM. Many mechanisms have been developed and published. In this thesis, RED and CoDel are presented.

2.2.2. Random Early Detection

Random Early Detection (RED) presents mechanisms for congestion avoidance in packet switched networks. RED was first described by Floyd and Jacobson (1993). The term congestion avoidance is used instead of the term congestion control, because the aim of RED is to avoid packet losses due to buffer overflows. RED also tries to keep the average sizes of queues at a low level. In this way, it tries to reduce queuing delays, but it also tries to allow occasional bursts of packets. RED is implemented only in routers and changes are not needed in the end-systems. It can also be deployed gradually. The router detects an incipient congestion with the help of the average queue size. This average queue size is calculated by using a low-pass filter. RED compares this average queue size to the predefined threshold values. If the average queue size exceeds a threshold value, senders are notified of incipient congestion with a certain probability. The router can notify senders either by dropping arriving packets or by setting congestion indicator bits. These indicator bits reside in packet headers. This bit setting is called “marking a packet”.

After the arrival of each packet, an RED router calculates the average queue size by using a low-pass filter with an exponential weighted moving average (EWMA):

$$avg(t+1) = (1-w) * avg(t) + w * q \qquad (2\text{-}1)$$

where avg is the average queue size, w is the smoothing factor, and q is the current size of the queue.

Due to the use of EWMA, short-term increases in queue size, which are the results of short bursts, have a minor effect on the average queue size. RED reacts only if the amount of traffic is increased in a long-term manner. The calculated average queue size is compared to two thresholds, a minimum threshold and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are dropped or marked. When the average queue size is greater than the maximum threshold, every arriving packet is dropped or marked. If packets are only marked, it is desirable that all source nodes are cooperative. This ensures that the average queue size does not significantly exceed the maximum threshold due to increasing traffic of non-cooperative connections.

Randomization arises when the average queue size falls between the minimum and maximum thresholds. Between the minimum and maximum thresholds, each arrived packet is marked or dropped with the probability of p, where p is a function of the average queue size:

$$p = p_{max} * \frac{(avg - \min\_th)}{(\max\_th - \min\_th)} \qquad (2\text{-}2)$$

where $Pmax$ = maximum dropping/marking probability (recommended value 0.1)

avg = average queue size

$\min\_th$ = minimum threshold

$\max\_th$ = maximum threshold.

This is depicted in a graphical way in Figure 2.2. RED avoids clustering of droppings, and that way eases error control, because the linearly growing probability function implements the relatively equal spacing of packet dropping or marking. Each time a packet is marked or dropped, the probability that this marked or dropped packet belongs to a particular connection is roughly proportional to the bandwidth share of this connection (Floyd and Fall 1999). Therefore, RED is able to punish the sources which generate congestion with their high sending rates.

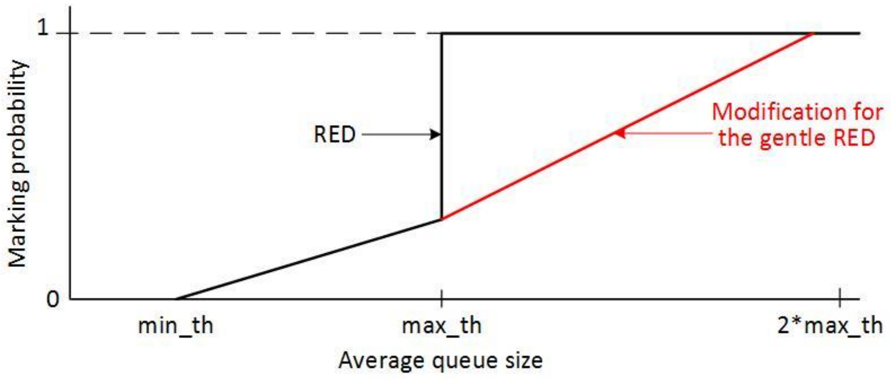


Figure 2.2 Marking/dropping probability of RED and gentle RED

RED has an extension called a “gentle” mode RED. The idea of the gentle mode RED (Floyd 2000) is to avoid the sudden jump of p to 1 after the average queue size exceeds the maximum threshold. To avoid this sudden jump, gentle RED slowly increases its drop rate as the average queue size approaches $2 * \max\_th$. This behavior of the gentle mode RED is depicted in Figure 2.2.

RED has four parameters to tune. Finding the optimal RED parameters is not an easy task (Christiansen et al. 2001). For example, the maximum threshold should be low enough to enable short delays. On the other hand, if the application is sending packets in bursts, fairly bursty traffic should be accommodated. With these types of applications, it could be desirable to have a large enough gap between the maximum threshold and minimum threshold. It can be observed that the optimal values for these parameters are connection-based. Unfortunately, it is impossible to use connection-based parameters because it leads to scalability and efficiency problems. However, RED also works quite well with suboptimal parameters, by reducing queuing delays and packet drops.

2.2.3. CoDel

CoDel (Controlled Delay) is a new active queue management mechanism (Nichols and Jacobson 2012). It tries to offer short delays while permitting bursts of traffic. Its purpose is to distinguish between a good queue and a bad queue. It treats these queue cases differently. A good queue allows short traffic bursts and is a situation which disappears within a short time. A bad queue persists for several RTTs. The robust way to separate these two cases is to take the minimum of the queue length over a sliding time window that is longer than the nominal RTT.

CoDel defines two parameters: the target parameter, which defines the target value for the long-term maximum queuing delay and the interval parameter, which defines the duration of one control period. There is not necessarily a need for the manual configuration of these parameters because default values can be used. The developers of CoDel have observed that a target of 5 ms and an interval of 100 ms are workable in most cases. The use of time values instead of packet counts makes the algorithm tolerant for varying link speeds. CoDel adds a timestamp to each received and queued packet. Once the packet has been dequeued, the time spent in the queue is calculated. CoDel is interested in the minimum delay over a time defined by the interval parameter. If the observed minimum delay is over the target parameter, it indicates a bad queue situation and CoDel starts to drop packets. Dropped packets are signals to the senders to reduce their sending rates, and the queue starts to drain.

CoDel is mainly used in edge routers and access nodes to control delays. It is able to offer small delays to QoS-aware traffic flows. In core routers, its use has not become more common because it increases the processing load of routers. In addition, most traffic flows can tolerate delays that are significantly higher than 5 ms.

2.3. Fairness

Fairness in computer networks concerns the sharing of network resources among traffic flows. Developing a fair congestion control has been considered, for example, by Denda et al. (2000) and Sivaraman et al. (2014). Simply speaking, fairness is achieved when network resources are divided in a fair manner. Unfortunately, it is often no easy task to define and implement fairness for computer networks. Fairness is an easy task as long as everybody divides the same resource and everybody has equal claims. As soon as these constraints are relaxed, things become difficult. Computer networks are very heterogeneous environments. There are different types of applications that send different amounts of data through different kinds of network paths.

Some preliminary studies exist to find out whether it is possible for a computer to "discover" the right rules for congestion control and fairness in heterogeneous and dynamic networks. The background is that there is no such congestion-control method that would be the best in all situations. Rather than manually formulating each endpoint's reaction to congestion signals, as in traditional protocols, congestion control algorithms for endpoints are derived in a more dynamic way. Winstein and Balakrishnan (2013) have considered the implementation of fair congestion control in this way.

In game theory, there are two concepts that are used in connection with fairness. The first concept is Pareto optimality (Srivastava et al. 2005). This describes a situation in which the profit of one party cannot be increased without reducing the profit of another. In practice, it means that resources are fully utilized. Unfortunately, it is difficult to say when the computer network is Pareto optimal. Delays increase when the network load approaches the capacity of the routers. Therefore, it is not possible to say where the point of the knee resides exactly. The second concept is Nash equilibrium (Srivastava et al. 2005). This states that if each player has chosen a strategy and no player can benefit by changing his or her strategy while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitute the Nash equilibrium. We can also say that a group of players are in Nash equilibrium if each one makes the best decision that he or she can make and, at the same time, he or she takes into account the decisions of the others. Therefore, the Nash equilibrium is some kind of balanced state which takes into account the other players. This kind of balanced state is unrealistic in computer networks because sending applications are dynamic and applications do not know each other's sending strategies.

On the Internet, a bottleneck refers to a router where the capacity of the connection path is the most limited. Perhaps the simplest and fairest way of dividing resources among users is to give the same sending rate to every user. In a computer network, this means that the sending rate is increased in an equal manner until the capacity of the first bottleneck is reached. After that, all users continue to send at that equal rate.

Unfortunately, this kind of approach is seldom Pareto optimal. If there are connections that do not pass through the first bottleneck, these connections can increase their rates without violating the other connections. The rates of the non-bottleneck connections can continue to increase one bottleneck after another until each connection has a bottleneck. This kind of rate allocation is called max-min fairness, and is defined in the paper by Cao and Zegura (1995). One definition for max-min fairness is that the allocation of rates is max-min fair if and only if every flow has a bottleneck.

Although it seems at first glance that the max-min allocation is reasonable, it must be remembered that operators wish to make a profit with their networks. The problem of max-min fairness is that it favors the connections of the first bottleneck router. In fact, its name comes from the property of favoring flows with smaller rates. These connections of the first bottleneck may be long-distance connections consuming resources in many routers, or they may be connections of a "lower paying level". The solution of this problem is the concept of proportional fairness. Proportional fairness is a compromise-based algorithm. This algorithm tries to maximize the total output of the network and, at the same time, it tries to allow at least a minimal level of service for all the users. The total output of the network can be throughput or the profit of the operator, for example.

The basic idea behind the profit of the operator is that every application or application type has its own utility function. This utility function describes the ability or willingness of the application to utilize extra bandwidth. It describes the willingness of the user to pay for the extra bandwidth. In Figure 2.3, based on the paper by Shenker (1995), the utility functions of two application types are shown. The first is a typical Internet application (web browsing, file transfer) called an elastic application. With this kind of application, after a certain point, the willingness for additional bandwidth decreases as the bandwidth granted to the user increases. The second is a so-called hard real-time application with constant bit rate sending.

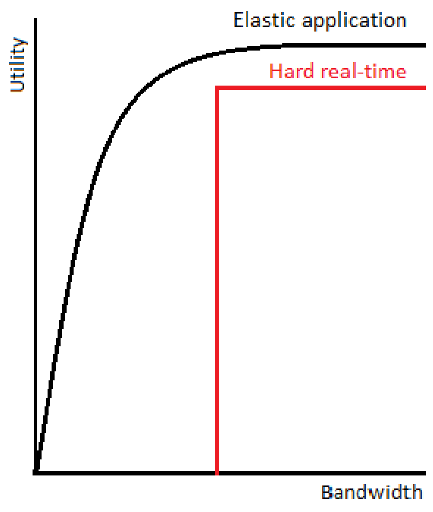


Figure 2.3 Utility functions

Finally, the resource allocation of a good fairness implementation can be defined. It should be Pareto optimal, proportionally fair, and satisfy the Nash equilibrium. In this case, the profit of the operator is maximized through the proportionally fair rate allocation based on utility functions and a fully utilized network (Pareto optimal). It is also desirable that the network is in a balanced state for at least some of the time (Nash equilibrium). Unfortunately, all these criteria are difficult, or even impossible, to achieve where computer networks are concerned.

The fair sharing of single network resources can be quantified by Raj Jain's fairness index (Jain et al. 1984). The fairness index is defined as:

$$f(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} {x_i}^2} \qquad \text{(2-3)}$$

where n is the number of connections and x_i is the rate allocation of the ith user. If $f(x)$ is 1, the rate allocation is perfectly equal. $F(x)$ becomes less than 1 upon the slightest deviation.

2.4. Round-trip times

The time duration between sending a certain packet and receiving the corresponding acknowledgement for that packet is called the round-trip time (RTT). The RTT can have noticeable effects on the congestion state of the network, and the efficiency of congestion control. Using the right RTT estimation plays an important role when considering the efficient use of the network. RTT *estimation* means that the last realized RTT measurement corresponds to the network state in the immediate past but we are often interested in the network state in the near future. For example, if RTT estimation is used to derive the retransmission timer (RTO) for the next packet under transmission, the latest RTT measurements are used with some kind of low-pass filter to perform RRT estimation. This RTT estimation is now an average from the history, and this average is used as a prediction for the future. Overestimating RTT causes less harm than underestimating it (Welzl 2005, p. 24). An RTT estimator should be fairly robust against short dips while ensuring appropriate reaction to significant peaks.

If the RTO is too long, it can take an unnecessarily long time until a packet is retransmitted. This can lead to inefficient behavior. Unfortunately, too short an RTO can also be a source of inefficiency. If the RTO is too short, unnecessary retransmissions can be caused and thereby network capacity wasted. The right timeout value is also important if a packet loss is interpreted as a signal of congestion. If the timeout value is too small, it can lead to false congestion detections, and thereby unnecessary rate reductions. If the timeout value is too long, it can lead to a situation in which the congestion is detected unnecessarily late. It is usually desirable for sources to reduce their rates as soon as possible if the network becomes congested.

Heterogeneous RTTs can cause some unequal fairness among traffic flows (Vojnovic et al. 2000) because traffic flows do not work necessarily in a synchronous way. For example, in an additive increase phase, where a sender increases its sending rate by a constant step for every ACK received, we can take the situation with two senders so that the RTT of sender 1 is twice as long as the RTT of sender 2. In this situation, sender 2 receives twice as many ACKs as sender 1 during the additive increase phase. Therefore, sender 2 also takes twice as many constant increase steps as sender 1. In this way, sender 2 can increase its sending rate faster than sender 1. The same conclusion can also be drawn if we consider the TCP throughput equation (Hassan and Jain 2004, p. 130):

$$T = \frac{s}{RTT} * \sqrt{\frac{3}{2p}} \; . \qquad \text{(2-4)}$$

This equation is the inverse square root p law, which is derived from the common saw-tooth behavior of TCP under the assumptions that the packet loss ratio p is constant and the receiver acknowledges every packet. This equation yields the average sending rate T in bytes per second as a function of RTT, the packet size s, and the packet loss ratio. It indicates that connections with shorter RTTs are given better average sending rates.

2.5. AIMD principle

In this section, the goodness of four basic congestion control approaches is examined. If the sending rate is changed with linear controls, the rate of the sender can be increased or decreased in an additive or a multiplicative way. There are four possible combinations:

- Multiplicative Increase, Additive Decrease (MIAD);
- Additive Increase, Additive Decrease (AIAD);
- Multiplicative Increase, Multiplicative Decrease (MIMD);
- Additive Increase, Multiplicative Decrease (AIMD).

These are not all the possible controls because our observations are restricted merely to the basic cases. For instance, the changing rate method may be exponential instead of linear, or both additive and multiplicative increase components may be present at the same time. The congestion feedback of the network may also be more complex than pure binary feedback (congestion, no congestion).

The behavior of these four cases with only two sources is depicted in Figure 2.4. This figure describes an ideal case. The RTTs of both senders are supposed to be equal and, consequently, the rates are changed at the same time. It is also assumed that both sources are given the same binary congestion feedback.

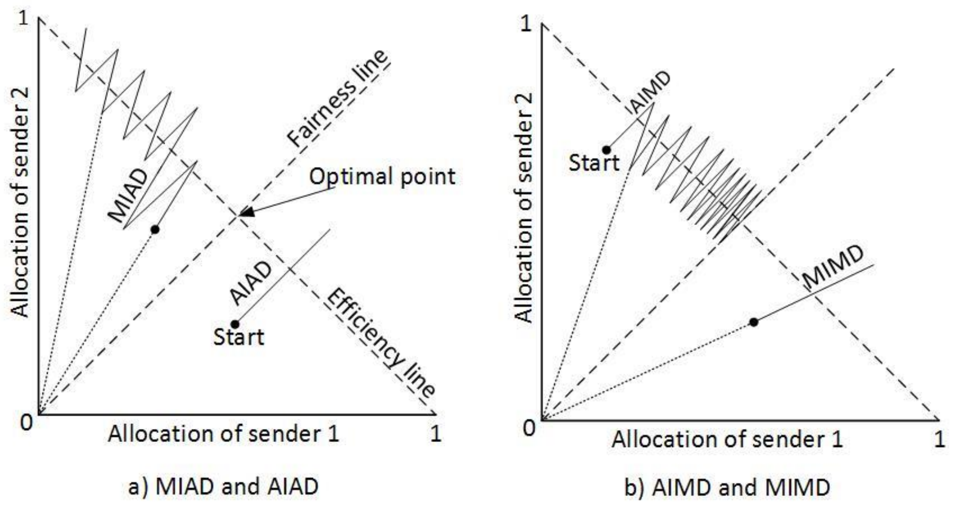


Figure 2.4 Transition lines of AIAD, MIAD, MIMD, and AIMD

There are two lines in the figure. The efficiency line shows the transmission capacity of the system. The sum of the sending rates cannot exceed this line for a long time. When the sum of the sending rates exceeds this line, the network will gradually become congested. The fairness line is at an angle of 45 degrees and passes through the origin. As far as fairness is concerned, the sending rates of the two customers should be equal. This is true for all points on this line. The optimal point is the point of intersection of the efficiency line and the fairness line. The optimal congestion control mechanism moves quickly towards this point and stays around it in a stable manner.

If there is an additive component, the transition line moves in parallel with the fairness line. It moves at an angle of 45 degrees. Therefore, AIAD cannot approach the fairness line unless the starting rates of the customers are equal. If there is a multiplicative component, the transition line moves along the line that dissects the origin and the point of the current sending rates. Due to this behavior, MIMD will never reach the optimal point if the starting point stays outside the fairness line. This is the same for the MIAD approach. If we wish MIAD to reach the optimal point, the starting point must be on the fairness line. Otherwise, MIAD behaves very unfairly. In contrast, AIMD always approaches the optimal point. The multiplicative decrease component of AIMD always brings the transition line closer to the fairness line, and the congestion control holds the total sum of sending rates near the efficiency line. However, the system fluctuates around the optimal point along the fairness line because only the binary feedback congestion indication is used. The senders always increase or decrease their rates after the feedback and the rates never stay in place.

In the paper by Chiu and Jain (1989), the above-described combinations are studied mathematically to prove the goodness of AIMD. They also present those very illustrative vector graphics representations that show that only AIMD can reach the optimal point. Chiu and Jain (1989) present three propositions. The first and third are as follows:

- Proposition 1. In order to satisfy the requirements of distributed convergence to efficiency and fairness without truncation, the linear decrease policy should be multiplicative and the linear increase policy should always have an additive component, and optionally it may have a multiplicative component with a coefficient of no less than one.
- Proposition 3. For both feasibility and optimal convergence to fairness, the increase policy should be additive and the decrease policy should be multiplicative (because a multiplicative increase component pushes the transition line away from the fairness line).

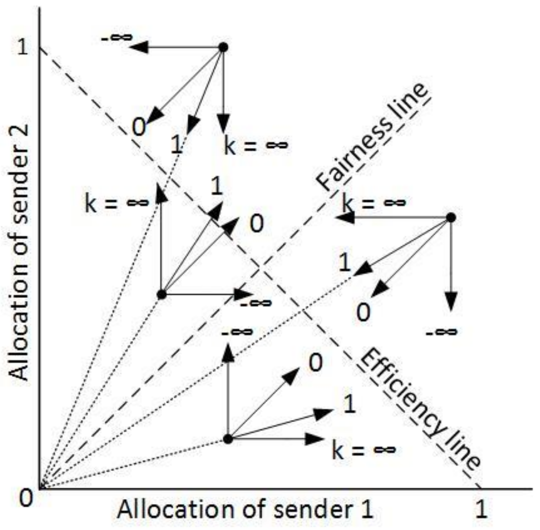
In view of these proposals, the congestion control mechanism should preferably be AIMD.

AIMD control can also be represented by a mathematical formula. If the rate of a sender at time t is denoted by $\mathrm{x(t)}$, and $\mathrm{y(t)}$ represents the binary feedback of congestion control so that the value of 0 means a non-congested situation and the value of 1 means a congested situation, the rate update function can be expressed as:

$$x(t+1) = a_i + b_i\, x(t) \quad \text{if } y(t) = 0 \qquad (2\text{-}5)$$
$$x(t+1) = a_d + b_d\, x(t) \quad \text{if } y(t) = 1 \qquad (2\text{-}6)$$

where a_i , b_i , a_d and b_d are constants and in the case of AIMD control $a_i > 0$, $b_i = 1$, $a_d = 0$, and $0 < b_d < 1$.

In the paper by Chiu and Jain (1989), the behavior of nonlinear controls has also been studied. They explain why they consider nonlinear controls to be unsuitable for practical purposes. Although nonlinear controls offer more flexibility in trying to move towards fairness, there is the problem of finding the right parameters. The parameters must usually be chosen relative to the system parameters, such as the capacity of links and the maximum number of users. Being too sensitive to the system parameters reduces the robustness of the control. A nonlinear control can also require different powers for different areas, as depicted in Figure 2.5.



Control function: new point = old point + nonlinear control

$x_i(t+1) = x_i(t) + a(x_i(t))^k$

Figure 2.5 Impact of the power k on the direction of the control

2.6. Stability, scalability, and selfish users

The fundamental properties of any congestion control algorithm are stability and scalability. Stability can be defined as a system's ability to remain in the steady state and avoid major oscillations due to changes in traffic conditions, at the same time as paying attention to fairness. Scalability refers to the capability of a system to achieve its desired behavior regardless of the system's size, which can be represented in terms of the amount of connections or capacity of links. These two properties are especially important with the Internet, which is a dynamic system under constant change.

Congestion control must work in a stable manner because the control mechanism is typically used in rather heterogeneous and uncontrolled environments. Users with different kinds of applications come and go. Jacobson (1988) presented the point of view that, in an equilibrium state, the connections can stay in the stable state when a new packet is *not* put into the network until the old packet leaves the network. This means that at any given time the integral of packet density around the sender-receiver loop is constant. If we apply this principle to the AIMD control shown in Figure 2.4, the additive component must increase the sending rate very slowly.

Concerning the Internet, the key element of scalability is the end-to-end argument first described by Saltzer et al. (1984). The idea behind this argument is that if we wish the mechanism to be scalable, the complex issues, such as packet retransmissions and security functions, should be placed at the network endpoints and preferably in the upper protocol layers. Thus, the implemented complexities do not burden the core routers. This is ultimately a very natural conclusion because functions can be implemented correctly only with the knowledge and support of the sending unit. Implementing the congestion control mechanism totally inside the core network is impossible because, with this kind of mechanism, there is no way to control sending rates. If the mechanism is partly implemented inside the core network, then the implemented mechanism should be offered for the use of wide variety of applications. It makes no sense to increase the processing load of routers by a mechanism that is used by only one application type. In addition, the mechanism should not require a lot of core router processing power. The applications-specific functions should be implemented completely by the endpoints.

Blumenthal and Clark (2001) presented that this end-to-end argument is no longer valid in all cases. For example, the original end-to-end paper by Saltzer et al. (1984) assume that endpoints are ready to cooperate to achieve their goals. Today, there are fewer and fewer reasons to believe that we can trust that all endpoints will behave as desired. If we wish to make networks more trustworthy, while endpoints cannot be trusted, it seems that some mechanisms should be located in the core of the network. In this way, endpoints can be forced into good behavior.

The congestion control mechanism with per-flow states on routers is not recommended because the per-flow state inside the router can cause scalability problems. If a router has to identify individual end-to-end flows, the router must maintain a database entry for each of such connections. These database entries must be updated continuously. For example, there could be a timer for each entry so that the entry can be cleared if the connection disappears without notification. This timer must be refreshed every time the packet of that entry is processed by the router. With IPv4, this kind of connection identification is also quite difficult because the port numbers of the transport layer must be investigated. This investigation is an impossible job if IPsec is used with IPv4 because port numbers are encrypted.

The congestion control implementation must have as much resistance as possible against selfish users. By selfish user, we mean a user who is ready to change the standard implementation of the congestion control algorithm so that improved performance is achieved by such a modified algorithm. This is typically realized through altering the code implementation of the control algorithm. It is very difficult to make the mechanism totally immune against selfish users as network security issues have shown during the last decades, but there is much that can be done. If the sender and receiver only receive a compiled version of the code, the code cannot be changed by a selfish user. In fact, this means that the congestion control is implemented as part of the application layer. If the implementation is part of the kernel, then open source operating systems, like Linux, are vulnerable to code changes. In such cases, implementation principles can be based on the presumption that the sender side is the trusted part, like the network operator. In these cases, the algorithm can be designed to be resistant against client side changes. One example of this kind of solution is presented in the next chapter when Explicit Congestion Notification (ECN) is explained.

2.7. Implicit and explicit congestion control

Congestion control can be implemented with or without feedback. In an open-loop control system, a pre-determined control strategy is used to define the fixed control, and feedbacks are not used. No measurements, conclusions, or adjustments are done during the control period. The length of this control period depends on the case. Because the Internet is vulnerable to routing changes, a fixed control strategy should perhaps be updated periodically. For instance, RSVP (Resource Reservation Protocol) (Braden et al. 1997) works in this updating manner.

In a feedback-based control system, the system is measured. Based on the measurements and a control algorithm, the appropriate feedbacks are derived and the control parameters of the system are adjusted. For example, if the target is to achieve a TCP-friendly throughput, a formula describing the throughput of a TCP session as a function of loss rate can be used as a control algorithm to calculate the estimated sending rate. This formula, published by Padhye et al. (1998), can be written as follows:

$$B(p) = \frac{1}{RTT\sqrt{\frac{2bp}{3}}+To\left(1.3\sqrt{\frac{3bp}{8}}\right)p(1+32p^2)} \qquad (2\text{-}7)$$

where:
$B(p)$ = the sending rate in packets per second
RTT = the round-trip time
To = the timeout value
p = the packet loss probability
b = the maximum number of packets acknowledged by a single acknowledgement.
This kind of feedback-based control system is sometimes called a closed-loop control system.

If we think of feedback-based congestion control in broad terms, two different kinds of approaches can be considered (Kurose and Ross 2017, p. 296; Jain 1990). The first approach is to perform congestion control without the explicit help of network elements. This kind of congestion control is called Implicit Congestion Control (ICC). It is also called End-to-End Congestion Control because the entire logic of congestion discovery resides in end systems. The second way to carry out congestion control is with the explicit help of routers. This kind of control is called Explicit Congestion Control (ECC) or network-assisted congestion control.

2.7.1. Implicit Congestion Control

In ICC, detection of congestion is based on the end-to-end behavior of traffic. One such behavior is a packets loss event since IP routers will discard incoming packets if there is no room in the queue due to the congestion. The second such kind of implicit feedback is an increase in end-to-end delay that is clearly over the fixed propagation delay. Congestion control using implicit signaling does not require any support from network nodes. One advantage of such kind of signaling is that new mechanisms are easy to deploy and the updates of existing implementations are also easy to execute. No changes are needed to the routers and therefore new mechanisms can be deployed gradually.

However, ICC also has some disadvantages. If packet drops are the only signs of congestion, and thus signals for network resources, the source must generate packet losses to know the limit of the network. Because the Internet is a very dynamic environment with incoming and outgoing traffic, this probing and generating of packet losses must be done repeatedly. This is the natural explanation for the saw-tooth behavior of the TCP protocol. Using only implicit feedbacks to make assumptions about the network state may be misleading. Interpreting packet losses as a sign of congestion works only if the congestion is the main reason for the packet losses (Tian et al. 2005). If a packet is dropped due to a checksum error or a temporary problem with a wireless connection, reducing the sending rate to alleviate the congestion is incorrect. In addition, if ICC is based on observations of delay variations, wireless connections can be problematic with their link-based retransmissions and handovers.

2.7.2. Explicit Congestion Control

The power of ECC is that it is exactly the congested node itself that reports that the node is congested. This significantly alleviates problems with dropped packets due to checksum errors and problems with wireless links. If ECC is done in the proper way, the effort for doing so is at an acceptable level. Only a couple of header bits are needed for ECC and the processing efforts of routers are minor. Therefore, it is no surprise that ECC has been adopted in the TCP/IP world by updates, as will be explained later.

ECC can be done in a backward or forward direction. In the backward-based approach, the congestion indication is sent straight to the source and it travels in the opposite direction to the congested traffic. In the forward-based approach, the indication is sent to the receiver typically with the packet that experienced the congestion. The receiver echoes this indication back to the sender. The advantage of the backward-based approach is that it typically takes less than 1/2 RTT to get the congestion indication to the sender. With the forward-based approach, this time is about one RTT, and it is, of course, preferable to react to congestion as quickly as possible.

However, the implementation of the forward-based approach is easier than that of the backward-based approach. In the forward-based approach, only the congestion indication bit must be set into the packet experiencing the congestion. With the backward-based approach, there are some problems. The first problem is the identification of the connection because part of the connection identifier resides in the transport layer. This transport layer part contains the port numbers of the source and destination applications. Examining transport layer headers violates the principle that routers only process the header of the IP layer. In addition, with IPsec, it is impossible to read the encrypted port numbers. The second problem is finding the packet of the same connection but traveling in the opposite direction to the congested packet so that the congestion indication can be piggybacked to the source. Of course, this backward indication can also be sent by a separate packet generated by the congested router. This kind of packet is called a choke packet. Both types of backward indication require extra processing from the router, and this occurs at a time when the router is overloaded due to congestion.

ECC can be binary-based, credit-based, or rate-based. In the binary-based approach, only one bit or a couple of bits are needed to report the congestion level of the network, In the simplest case, only one bit reports whether the network is congested or not. However, with this kind of one-bit implementation, more control bits are needed for proper implementation as we will see in the next chapter when the ECC implementation of TCP/IP is considered. In the credit-based approach, the source is told how many bytes or packets the source is allowed to send until the next new credit is received. This is quite a similar mechanism to receiver-based flow control. If the rate-based approach is used, the sender is told the exact sending rate with packets per second or bits per second.

If we compare these three approaches, it can be said that the binary-based approach enjoys the widest acceptance. Only monitoring the length of queues is necessary to conclude the proper congestion feedback. The credit-based and rate-based approaches need more processing efforts from routers. Used and free resources must be investigated and the number of active connections using resources must be monitored to calculate the explicit credit or rate for a certain user. Monitoring the number of connections is also problematic without violating the protocol approach because port numbers reside in the headers of the transport layer. As stated above, these port numbers are impossible to read when using IPsec. The binary-based approach also has its own problems with special cases. One such case is using IP tunneling for instance with Mobile IP or VPN. The ECC feedback must be copied from the tunneling header to the original header at the endpoint of the tunneling.

2.8. Window-based versus rate-based congestion control

There are two basic ways to control the sending rate of the sender. The first is rate-based control and the second is window-based control (Chandra and Subramani 2010). In rate-based control, the sender is not allowed to exceed a certain rate, which is usually presented in bits per second. The sender is allowed to send continuously using this rate until the next congestion feedback is received. After receiving the congestion feedback, the sending rate is updated according to certain rules. In window-based control, the sender is allowed to send a certain maximum number of bytes or packets to the network between congestion feedbacks. This number of bytes or packets is called a sending window. This method can give rise to burst sending behavior because the sending window may become empty before the next feedback arrives. Window-based control is said to be self-clocking (Jacobson 1988) because the behavior of the window is strictly coupled to the arrivals or absences of feedback messages, as well as to the cycles of feedback messages.

The window-based method works in a more network-friendly way. It naturally implements the principle that a new packet is not put into the network until the old packet leaves the network. It automatically stops the sending function if the sender does not receive any feedback messages due to a major problem somewhere in the round-trip path. On the other hand, the window-based method has the disadvantage that if the round-trip delay is long, the network can sometimes become underutilized because it takes a long period before the self-clocking feedback system reaches the optimal transmission rate. In addition, if feedback messages arrive in a bursty manner and the sender ejects packets into the network as soon as possible, the traffic pattern can be very bursty. For instance, this kind of feedback burstiness can appear if frequent acknowledgement messages are queued in a sequential manner due to congestion in a high-speed router.

Rate-based implementation is typically simpler than the window-based one. Rate-based control is also said to be more suitable for streaming media because it does not stop the sending if feedback messages stop arriving for a while. Akan (2004) has analyzed the throughput of rate-based and window-based congestion control schemes. One important result of that analysis is that the throughput performance of the rate-based congestion control schemes is inversely proportional to the square root of the propagation delay, whereas that of the window-based schemes is known to be inversely proportional to the propagation delay itself. This result confirms the point of view that rate-based congestion control schemes adapt better to high propagation delay environments than window-based schemes.

2.9. Overprovisioning

Nowadays, the common choice of ISPs is to use overprovisioning to avoid congestion in their networks. Overprovisioning is commonly used by operators to protect their networks against unexpected traffic variations. It typically involves dimensioning network links and the capacity of routers so that these resources exceed the expected traffic load by a certain margin. In other words, the bandwidth B of the link is chosen so that:

$$B > (1 + F) * E \qquad (2\text{-}8)$$

where E is the expected base offered traffic of the link and F is the overprovisioning factor. Through overprovisioning, operators wish to ensure that their networks can absorb both expected and unexpected traffic load.

In theory, the challenge is to determine the right overprovisioning factor. This factor is necessary in order to offer the desired level of protection against a given range of traffic surges. Because there is a need to accommodate relatively large traffic fluctuations caused by link failures, Huang and Guerin (2005) note that values of the overprovisioning factor of about five or even higher are not uncommon in large IP backbones. Huang and Guerin (2005) state that the emergence of high-speed applications and access links, and the greater heterogeneity of users, mean that there is considerable uncertainty regarding whether even such conservative overprovisioning factors can remain adequate. By using better congestion control mechanisms inside the network, we can suppose that it possible to use smaller overprovisioning factors.

The reasons for overprovisioning are mainly financial. With overprovisioning, the operator is ready for future growth of traffic load and number of customers. Operators have made investments for future growth. The operators are ready to receive new customers as well as new traffic from old customers within a short time. If an operator has a network without overprovisioning, and thus only just enough bandwidth to serve the expected maximum load, then there is an increased risk of network failure. Network failures are harmful for operators because they lead to customer complaints. These failures also generate extra work for network administrators. If the excess bandwidth costs less than the amount of money that an ISP could expect to lose through customer complaints and increased administration costs, overprovisioning is worth implementing. This is currently the situation because bandwidth has become low-cost.

However, overprovisioning does not render proper congestion control mechanisms unnecessary. Internet connections often travel through a number of heterogeneous networks. There may be some operators along the connection path that do not implement major overprovisioning inside their networks. These lightly overprovisioned networks can become congested. Link failures can also increase the load to the congested level when all traffic from the faulty link is moved to the operational part of the network. The self-similar nature of Internet traffic can also cause congestion in an overprovisioned network.

A phenomenon is considered self-similar if it behaves in the same manner on different scales in a certain dimension (Sahinoglu and Tekinay 1999). In the case of network traffic, this dimension is time. In a traffic pattern, self-similarity is seen as the presence of bursts of different lengths. There are bursts on every time scale, ranging from a few milliseconds to minutes and hours.

Self-similar characteristics in network traffic present a different set of problems for network design compared with the traditional Poisson type traffic. With Poisson type traffic, short-term fluctuations average out and the rate over a long timescale approaches a constant value. In terms of congestion control, queues behave in a different way than with Poisson type traffic. The more self-similar the traffic is, the longer the queue sizes should be. The queue lengths of self-similar traffic decay more slowly than with Poisson sources. Therefore, self-similar traffic places the network in danger of persistent congestion, which has a negative impact on network performance. Poisson processes behave well because peak loading is not sustained, and so queues do not tend to fill up. With self-similar type traffic, congestion control is more important than with Poisson type traffic.

A number of studies have shown that VBR (Variable-Bit-Rate) video transmitted over the Internet is self-similar. Garret and Willinger (1993) introduce a detailed statistical analysis of a two-hour-long sample of VBR video. They found that the sample could be modeled using self-similar processes. Rikli (2011) found that all the tested VBR video traces could be considered self-similar.

With proper congestion control, the use of unnecessarily large overprovisioning can be avoided. Avoiding unnecessarily large overprovisioning promotes green Internet thinking (Gupta and Singh 2003). Green Internet thinking means that we wish to develop and adopt new and better techniques so that the Internet can be used in a more energy-efficient way. During the next ten to fifteen years, the Internet is expected to undergo remarkable changes with respect to the number of users, traffic amount, and types of applications. Due to these changes, the energy consumption of the Internet is expected to grow rapidly.

2.10. Challenging environments

Congestion control has to work in very heterogeneous environments. Network users have many types of end devices. There are conventional desktop computers with a stable location and portable mobile phones. There are also many different types of transmission links, some with special characteristics. There are wireless links with certain probability of high bit error rates, links with high delays, and asymmetric links with low-speed feedback channels. Sometimes congestion-controlled traffic must pass through a multicast network environment or travel through special boxes like firewalls and NAT boxes. These environments are briefly discussed here, concentrating on the challenges they place on congestion control. The presentation shows that developing a congestion control mechanism suitable for all cases is extremely difficult.

Wireless links can be harmful for congestion control (Casetti et al. 2002) because congestion control implementations typically use delays and packet losses as an indication of congestion. Wireless links with high bit error rates can affect both of these variables in unexpected ways. If retransmissions are not used, packet losses can be quite usual. If retransmissions are used, sequential retransmissions can increase delays and especially variation in delays. Many efforts have been made to solve this problem. Many of these efforts have especially targeted the TCP protocol. Typically, solutions try to hide the problems of wireless links from the end-to-end connection. There is a splitting solution, dividing the connection into the wireless part and the wired part (Bakre and Badrinath 1995) and a snooping solution with quick transport layer retransmissions (Balakrishnan et al. 1995). These solutions typically have some stumbling block. For example, there may be problems with security issues, especially with IPsec, which encrypts the header structures of the transport layer. It seems that the natural and working solution is to solve these problems totally on the data link layer using techniques like Forward Error Correction (FEC), retransmissions, and varying link layer packet sizes.

Satellite links share the same inclination to high bit error rates as wireless links. Because of their long signal paths, satellite links also have some extra problems (Katabi et al. 2002). First, the delays are usually long, and also RTT estimates. This typically means that RTT fluctuations are at a high level, which is normally an undesirable feature. Satellite links are often high-speed links and, together with high delays, these links have high bandwidth-delay products. Connections having high bandwidth-delay links are problematic for an AIMD control mechanism. It takes a long time before moderate additive increase steps can reach the high utilization level of the connection path. After the congestion, the multiplicative decrease component again takes the connection path far away from the high utility level. This problem can be alleviated if this kind of link is shared among many AIMD connections.

In asymmetric links, the offered data rate of a reverse path is smaller than that of a forward path. Typically, the forward path is the same as the data path and the reverse path is the same as the acknowledgement path. This kind of asymmetric phenomenon can appear if the connection path includes for example slow ADSL or satellite links. This asymmetry can also occur if the shared data channels of the mobile phone networks are used in the opposite manner. This means that the heavy loaded downlink forms an acknowledgement path and the moderate loaded uplink is now used as a data path. In asymmetric situations, problems can appear if the reception of acknowledgements or the round-trip delays define the sending rate of the forward path. If the acknowledgements do not arrive at the sender at the expected frequency, the data-sending rate can be decreased even if the forward path is in a congestion-free state. This can lead to the under-utilization of the available bandwidth (Fu et al. 2001).

Multicast improves the efficiency of multipoint video distribution by building a distribution tree from the sender to a set of receivers. Several multicast congestion control protocols have been proposed (Matrawy and Lambadaris 2003; Yang and Lam 2006). These protocols can be classified into approaches using single-rate sending and approaches using multi-rate sending (Chandra and Subramani 2010). The single-rate approach means sending to the whole group using only one rate. This rate can be changed during the connection by congestion control. The multi-rate sending approach can be either replicated or layered. Under the replicated approach, the sender sends the stream using a few rate classes. The receivers are partitioned into groups and each receiver joins a certain class. In the layered approach, the data stream is organized in a layered way. The receiver incrementally joins higher groups according to its available bandwidth.

There are three problems that have to be solved by the multicast congestion control protocol (Welzl 2005, p. 42; Yang and Lam 2006). These problems concern mainly the single-rate approaches, but partially also the multi-rate approaches. These problems are described below with some proposed solutions.

The first problem facing multicast congestion control is that of feedback implosion. If a large number of multicast receivers independently send congestion feedback to the multicast sender, the cumulative amount of such signaling traffic increases as it travels up in the multicast tree. The link near the multicast sender can become congested because of this cumulative congestion feedback traffic. The same feedback implosion problem occurs if receivers send ACKs or NACKs to implement reliable multicast communication. There are two approaches to alleviate this feedback implosion problem. If a suppression-based approach is used, not all receivers will send their feedback to the sender. Only some receivers, which are chosen as representatives, are allowed to send congestion feedback to the sender. In an aggregation-based approach, receivers do not send their feedback directly to the sender. Instead, they send the feedback up to some intermediate node. This node uses the information of multiple feedback messages to calculate the content of a single collective message.

The second problem concerns congestion indicator filtering. Unlike a unicast sender, which just needs to control a single connection between a sender and a receiver, multicast congestion control has to deal with a wide range of connection paths for each receiver. These independent receivers can experience a wide variety of quality properties. A single packet may have been lost or delayed along the way to some receivers, but may have been successfully received by most of the others. As the number of receivers increases, the range of possible congestion feedbacks increases. Therefore, some kind of filtering is needed. The type of filtering used depends on the desired result. If we do not want a single receiver to experience intolerable quality, we must choose the most dis-serviced receiver as the representative feedback source. If we want to provide good quality on average, we must take into consideration the feedback of every receiver. In this case, however, some kind of filtering must be done. In a suppression-based approach, this filtering can be done by allowing, in a certain period, the feedback of only a few receivers, and changing the representative receivers in a round robin fashion. In this case, it is preferred to change the representative receivers in some random way to avoid the phase effect. In an aggregation-based approach, this filtering can automatically be done by intermediate nodes.

The third difficulty of multicast congestion control is the fairness problem. Fairness is a complicated issue in unicast traffic. With multicast, fairness is presumably even more complicated. Over the years, one challenge has been the lack of an agreed definition for multicast fairness. For example, if the aim is to work in a TCP-friendly way, is the comparison done using an average or using the slowest bottleneck node?

An important principle behind the continuous development of the Internet has been: be conservative in what you do and liberal in what you accept from others. Being conservative means that the protocol only does what it should do and nothing else. To put it simply, liberal protocol implementation means that the protocol also accepts unknown behavior from a peer entity. This is done by ignoring unknown behavior. This liberal behavior also makes it possible to deploy new mechanisms gradually. The latter principle is a good starting point when developing a new congestion control mechanism for the Internet. However, there is one element in the network that does not behave in a liberal way. This element is the firewall. We have to ensure that the new mechanism also works with firewalls.

3 TCP/IP Congestion Control and Video Transfer

In its early years, the TCP/IP protocol suite did not have any kind of congestion control mechanism. Therefore, it was unsurprising that TCP/IP networks experienced their first congestion collapse events in the 1980s. After these collapse events, something had to be done. The natural choice was to implement TCP/IP congestion control as part of the TCP protocol because most of the traffic was TCP-based at that time. If the traffic was not TCP-based, it was typically short-lived traffic, and therefore normally not the source of congestion. The TCP-based solution was implemented in a very clever way. Only the code of the sender side had to be changed without any need for changes on the receiver side. Receivers continued to send only normal acknowledgements. Based on these acknowledgements, the senders made implicit assumptions on the state of the network. Because congestion control was implemented in this way, it was possible to deploy it in a gradual way. Even the later improvements could be made mainly without any changes to the receiver side.

UDP-based traffic with long-lived connections also increased gradually. There were assumptions that this kind of traffic could be harmful without congestion control. Therefore, a period of time started when there was a lot of work aiming at developing congestion control mechanisms for UDP-based traffic. The most notable solution was the DCCP protocol explained later in this thesis. Because the developed mechanisms did not enjoy wide popularity among network operators, this kind of research work gradually decreased. However, during recent years, all kinds of video-based traffic has increased heavily. People want to watch TV programs through the Internet. All kinds of events, including sport events and music concerts, are watched via the Internet in real time. So, it can be assumed that congestion control for video services will become a active topic of research. This thesis can be categorized as belonging to this research category.

3.1. TCP congestion control

In its basic mode, TCP uses an implicit congestion control scheme. TCP increases its sending rate until a packet loss is detected and, after that, the sending rate is reduced. The reducing operation is multiplicative and large enough to pull the network out of the congested state. After the rate reduction, the sending rate starts to increase again. This pattern of increasing and decreasing the sending rate makes the data flow fluctuate around the equilibrium point. Load and connection changes inside the network change the position of the equilibrium point, but the multiplicative decrease scheme always returns the data rate to below the equilibrium point. In fact, the choice of the right decreasing step for the multiplicative decrease scheme is the key to the good performance of the AIMD algorithm.

TCP congestion control follows the ACK clock principle. In its steady state, the network cannot become congested as long as a new packet is not sent to the network before the previous packet has left the network. The acknowledgement advertises that the previous packet has left the network. In this way, the arrival of ACKs can synchronize the sending behavior of the sender.

3.1.1. TCP timeout management

TCP maintains an estimate for the round-trip time (RTT) to derive a retransmission timer for each transmitted packet. Because the delay properties of TCP connections are variable, the RTT estimate is derived by a dynamic algorithm that constantly adapts to the changes of a connection path. TCP implementations attempt to predict future round-trip times by measuring the delay properties of sent packets. Based on these measurements, the sender averages these samples into a smoothed round-trip time estimate (SRTT). When a packet is sent over a TCP connection, TCP measures how long it takes for it to be acknowledged. Consecutive measurements produce a sequence of round-trip samples: $S(1)$, $S(2)$, $S(3)$ and so on. After each new sample $S(i)$, the new SRTT is computed as:

$$SRTT(i+1) = (1-\alpha) * SRTT(i) + \alpha * S(i)) \qquad (3\text{-}1)$$

where α is the smoothing factor that determines how quickly SRTT adapts to changes and how quickly old values are forgotten. It is recommended to use α value of 1/8. A formula like this is an Exponentially Weighted Moving Average (EWMA) or a low-pass filter that helps to change SRTT smoothly.

Choosing a suitable retransmission timeout (RTO) value based on SRTT is no trivial task. The initial implementations of TCP used the rule:

$$RTO = 2 * SRTT. \qquad (3\text{-}2)$$

However, experience has shown that a constant factor is too inflexible because RTT and its variations increase quickly with load. Jacobson (1988) gives a concrete example of 75% capacity usage, leading to an RTT variation factor of sixteen. He also notes that an SRTT factor of 2 can adapt to a load level of at most 30%. This problem was solved by making the timeout value sensitive to the variance of the round-trip time as well as using a smoothed round-trip time estimate. This change requires keeping track of another smoothed variable, the RTTVAR (Round-Trip Time Variation), which is updated using the formula:

$$RTTVAR(i+1) = (1-\beta) * RTTVAR(i) + \beta * |SRTT(i) - S(i)|. \quad (3\text{-}3)$$

This is also EWMA and β of ¼ is suggested. The retransmission timeout is set according to:

$$RTO = SRTT + 4 * RTTVAR. \quad (3\text{-}4)$$

The choice of factor 4 is somewhat arbitrary but multiplication by 4 can be done by a single shift.

Making these RTT measurements is a hard task for the TCP entity because there are typically many open connections at the server side at the same time. Therefore, TCP implementations have traditionally had only one RTT measurement open per connection at a time, leading to one measurement per RTT. Some research work has been done to try to find out if one measurement per RTT is sufficient. When using fairly modest congestion window sizes, it is suggested that timing each segment does not lead to a better RTT estimator (Paxson et al. 2011). Instead, Jacobson et al. (1992) suggest that TCP connections, which utilize large congestion windows, should take many RTT samples per window of data to avoid aliasing effects in the estimated RTT. For example, it is possible to use each ACK as an RTT sample if the timestamp option is included in the TCP header.

TCP uses Karn's algorithm (Karn and Partridge 1987) when taking RTT samples. This algorithm is needed because TCP cannot distinguish between ACKs corresponding to regular segments and ACKs corresponding to their retransmissions. The algorithm is simple and notes that do not update the estimates after receiving the acknowledgement whose acknowledgement number corresponds to the retransmitted segment. Karn's timer back-off algorithm is also used. This algorithm doubles the RTO value of a certain segment after the RTO timer of this segment expires. This back-off behavior is regarded as the last attempt to leave the congested state.

3.1.2. Increasing sending rate by using slow start and congestion avoidance algorithms

TCP’s slow start (Allman et al. 2009) is initiated at the start phase of the TCP connection and after a badly congested state (after a retransmission timeout). The slow start has two targets. The first target is to prevent a new connection from using a high initial rate at the beginning of the connection. By using a high initial rate, the just-opened TCP connection could drive an almost congested network into a badly congested state. With the help of the slow start algorithm, the TCP sender can probe the available bandwidth by increasing the amount of data injected into the network. This behavior prevents congestion of the network with a large burst of data. The second target is to reach a reasonable sending rate quickly enough. If the sending rate is increased too slowly, the network can be underutilized for a long time after a massive rate reduction. Short-lived connections can never reach high sending rates. These two targets are satisfied by using a low initial rate but increasing the sending rate in an exponential manner. Because the rate-increasing manner is actually exponential, the term slow start is somewhat misleading.

TCP does not actually define its congestion window in units of a number of segments. TCP is a byte-oriented protocol and the congestion window is also measured in bytes. However, to simplify matters, the number of segments is often used as the unit for the length of the congestion window. Thus, the slow start algorithm can be defined briefly as: *for every acknowledgment received, increase the congestion window by one segmen*t. At the beginning of the connection, only one segment may be sent and a low initial rate is used. Once the acknowledgement is received, the congestion window can be increased from one segment to two segments. Two segments can now be sent. When both of these two segments are acknowledged, the congestion window can be increased to four segments. As can be seen, the congestion window increases exponentially. The sender also behaves in a burst-like way. The allowed number of segments is normally sent as quickly as possible. After that, the sender will stop to wait for the acknowledgements.

Over the decades, transmission rates have increased and connection paths have become longer. Initiating slow start with only a single segment is inefficient. Therefore, Allman et al. (2002) made it possible to use a bigger value for the initial window. They specify the upper limit for the initial window as:

$$Initial\ Window = \min(4 * MSS, \max(2 * MSS, 4380\ bytes)). \qquad (3\text{-}5)$$

where MSS is the connection-based value that specifies the largest amount of application data for a single TCP segment. For example, the wired Ethernet technique adheres to the maximum frame size of 1500 bytes. Sending a 1500-byte Ethernet frame indicates the maximum segment size of 1460 bytes (IP and TCP headers both take 20 bytes). In this case, the initial window is 4380 bytes and the sender is allowed to send up to three segments at the beginning of the slow start phase. Dukkipati et al. (2010) propose increasing TCP’s initial congestion window to at least ten segments. In addition, Touch (2012) proposes that TCP can objectively measure when an initial window is too large, and that measured value could be used over long timescales to adjust the initial window automatically.

The exponential growth used in the slow start phase cannot continue endlessly. TCP has a parameter called a slow start threshold (ss_threshold), which ends the slow start phase.

This parameter is often set to the value of 64 KB at the beginning of connections, but other initial values are also possible. During the connection, the value of this parameter is derived from the value of the congestion window in a manner that will be explained later in this thesis. After the ss_threshold is reached, the connection switches to its linear increase mode. This linear growth phase is called the congestion avoidance phase.

In the congestion avoidance phase, instead of increasing the congestion window by one segment for each received acknowledgement, the algorithm increases the congestion window ($cwnd$) using the equation:

$$cwnd(i+1) = cwnd(i) + MSS * MSS/cwnd(i). \qquad (3\text{-}6)$$

In this equation, both $cwnd$ and MSS are presented in bytes. If we take the normal assumption that it takes about one RTT to send the whole window, this means that the window is increased by one segment per round-trip time. This is the additive increase part of TCP congestion control.

The sending rate increases very fast at the final part of the slow start phase. The doubling of the window is sometimes too aggressive and it is possible that the size of the congestion window grows far larger than the actual bandwidth of the connection path. This can cause multiple packet losses. Wang et al. (1998) presented a modification to the slow start algorithm called smooth start, which switches from exponential growth to linear growth in a smooth way. In smooth start, the value of ss_threshold/2 is used as a separator to divide the original slow start phase into two phases. If the congestion window size is less than ss_threshold/2, the sender is in the first sub-phase, which is called the filling phase. In this sub-phase, the smooth start algorithm behaves in the same way as the traditional slow start. This means that the size of the congestion window increases by one segment for each received ACK. If the window is above the value of ss_threshold/2, the sender is in the second sub-phase, which is called the probing phase. In this phase, the congestion window size increases more slowly and multiple ACKs are needed to increase the congestion window by one segment. This is presented in Figure 3.1.

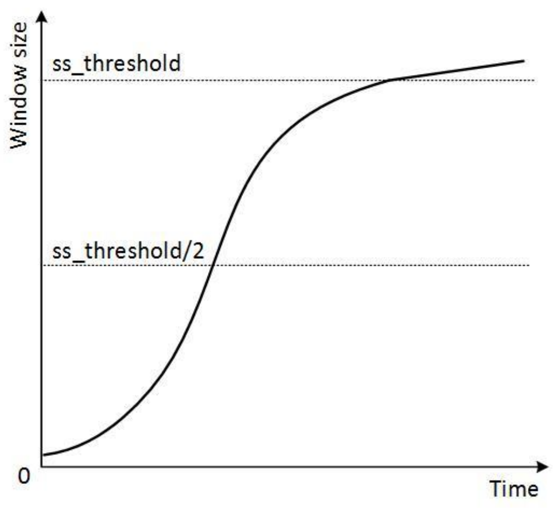


Figure 3.1 Illustration of the smooth start algorithm: in the first phase the congestion window size grows exponentially up to ss_threshold/2, slower than exponentially in the second phase, and then linearly in the congestion avoidance phase.

However, TCP connections do not always send at the rate of the congestion window. TCP’s sending window is the minimum of the congestion window and the receiver's advertised window. The receiver’s advertised window is based on flow control. With the help of flow control, the receiver can tell the sender how much free room there is in the receiving buffer. It is also possible that the TCP protocol does not have enough data to transmit because the application layer cannot push enough data to the transport layer. The network interface can also restrict the sending rate.

3.1.3. Decreasing sending rate by using slow start, fast retransmit, and fast recovery algorithms

If the RTO timer of the segment expires, it is an indication to the TCP protocol that there is a bad congestion state inside the network. After observing this bad congestion state, TCP decreases its sending rate in a radical way. This is done using the slow start algorithm. The congestion window is set to its initial value (one segment or a few segments). At the same time as initiating the slow start, the ss_threshold parameter is set to the value of half of the congestion window just before the timeout operation. After the timeout operation, the slow start phase continues until this new ss_threshold value is reached. After that, TCP enters the congestion avoidance phase. When the RTO timer expires, the unacknowledged segment is retransmitted and the RTO value of that segment is doubled. This doubling also alleviates the bad situation slightly.

The aim of the fast retransmit algorithm is to trigger a retransmission before the RTO timer of the segment expires. In the case of a moderate congestion situation, the slow start operation can be prevented after the fast retransmit operation by activating the fast recovery algorithm. Fast retransmit takes advantage of the earlier defined rule for TCP behavior. The rule defines that if a TCP entity receives an out-of-order segment, it must immediately send an ACK frame with the acknowledgement number representing the last in-order segment received. This acknowledgement is called a duplicate ACK (DupACK) because it has the same ACK number as the last sent acknowledgement. TCP will continue to repeat this duplicate ACK for each out-of-order received segment until the missing segment arrives and fills the hole in the receiving buffer. After receiving the missing segment, the receiver sends a cumulative acknowledgement for all the in-order received segments.

When the sender receives the duplicate ACK, it can mean two different types of events. In the first case, the packets have been reordered inside the network. This means that consecutive packets have taken different routing paths between the end nodes. In the second case, the packet has been lost on the network path. On the basis of this second case, duplicate ACKs can be considered an early warning from the system to report that a packet has been lost. This is an early warning because often the RTO has not yet expired. Therefore, with the help of the duplicate ACKs, the sender can initiate the fast retransmit operation for the missing segment even though the RTO timer of the segment has not yet expired. In fact, the sender must receive three duplicate ACKs with the same acknowledgment number before the missing segment can be fast retransmitted. Three duplicate ACKs are needed instead of one because we need to be fairly sure that it is not a reordering situation.

Related to congestion control, the more interesting algorithm is the fast recovery algorithm that is always initiated after the fast retransmit operation. Because the sender receives duplicate ACKs, it is a clear indication that at least some packets have found their way to the receiver through the connection path. Therefore, the congestion situation cannot be very critical. There is probably a moderate congestion situation somewhere on the connection path and it is not a good idea to react to moderate congestion with the slow start operation. Slow start drops the sending rate to almost zero and it takes a long time before the pipe is again utilized at a high level. Therefore, this kind of moderate congestion activates the fast recovery operation, which only halves the sending rate.

Fast recovery is implemented as follows:

- When three duplicate ACKs are received and fast retransmit is triggered, the ss_threshold is set to half of the congestion window just before the congestion event. After that, the fast recovery algorithm governs the transmission of data as explained below until a non-duplicate ACK arrives.
- Set the congestion window to the value of ss_threshold plus $3 * MSS$. This increases the congestion window by the number of the three segments that have just left the network and initiated the three duplicate ACKs. These three segments are buffered on the receiver side waiting for the missing segment.

- For each received additional duplicate ACK, the algorithm increases the congestion window by MSS. This artificially inflates the congestion window to reflect the extra segment that has left the network.
- Transmit a new segment if allowed by the new value of the congestion window. Note that sometimes the sender must stop sending for the duration of several duplicate ACKs. The algorithm does this because there may be a lot of un-acknowledged data out in the network and the congestion window has just been halved. Such an example is given in the paper by Hoe (1996).
- When the next ACK arrives so that it acknowledges new data (this is usually the cumulative acknowledgement that acknowledges the missing segment plus the other later segments), the algorithm sets the congestion window to the value of the ss_threshold. This ACK is the non-duplicate ACK that ends the fast recovery algorithm.

After the fast recovery phase, the congestion avoidance algorithm, with the normal additive increase behavior, takes over and starts from the latest setting of the ss_threshold.

In summary, Figure 3.2 presents the different phases of TCP's congestion control. Packet loss is discovered at the time unit of 8. The difference between the fast recovery algorithm (3 DupACKs) and the timeout-based slow start is presented immediately after this loss event. The fast recovery algorithm halves the sending rate. Slow start drops the sending rate to almost zero.

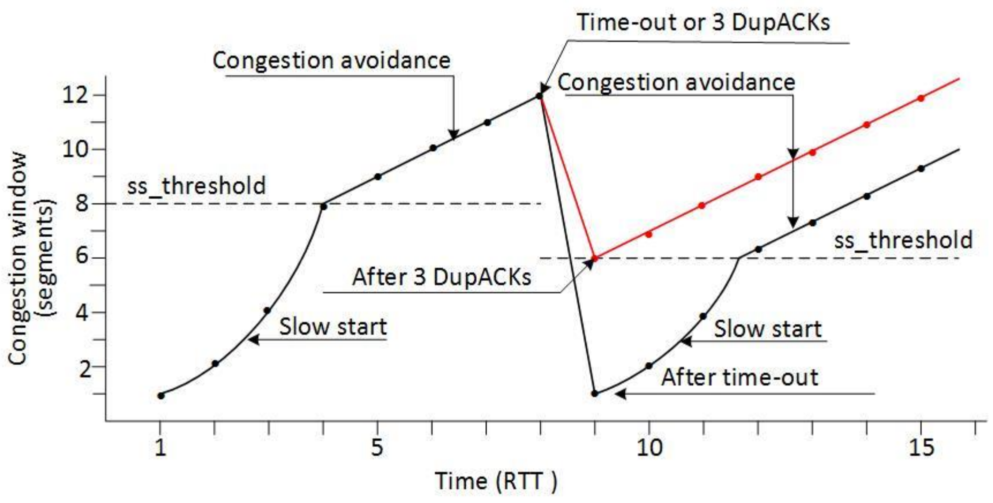


Figure 3.2 TCP's congestion control

3.1.4. TCP versions

TCP did not have any kind of congestion control in its very early years. A lot of development work has been needed to add congestion control to the TCP protocol afterwards. This development work has generated many TCP versions, each having its own name. The first version was named TCP Tahoe. Its congestion control implementation is as described above except for fast recovery and, therefore, TCP Tahoe is based on the slow start, congestion avoidance, and fast retransmit algorithms. The next version was called TCP Reno. Reno improves the congestion recovery phase by adding the fast recovery algorithm to the TCP congestion control.

One of the known problems of the fast retransmit/fast recovery algorithm is a situation when multiple packets are lost during a single window of data. The problem is that the fast retransmit operation initiates the retransmission of a single segment. This means that TCP retransmits only one segment per round-trip time. After this retransmission, it takes one RTT until the next non-DupACK arrives. This regular ACK includes information regarding which segments have actually made their way to the receiver. If more than one segment has been lost during a single window, this regular ACK is known as a partial ACK. A partial ACK is a regular ACK that covers some, but not all, of the segments that were sent before entering the fast retransmit/fast recovery phase (Henderson et al. 2012). After this partial ACK, three new DupACKs are needed to initiate the fast retransmit operation for the next missing segment. After this new fast retransmit operation, a new fast recovery operation is triggered and the congestion window is reduced again due to the condition inside a single window. Unfortunately, it is also possible that this behaviour, with its many RTTs, may cause the expiration of the RTO timer. In this case, the sender must enter its slow start mode and radically reduce its sending rate.

One solution for the problem of multiple drops within a single window is a TCP version called NewReno (Henderson et al. 2012). Every time the fast retransmit operation is activated, TCP NewReno stores the highest sequence number so far transmitted to the variable called *recover*. With the help of this variable, the sender can distinguish between the full ACK and the partial ACK. If the next regular ACK after the fast retransmit operation is the partial ACK, the next missing segment is immediately retransmitted without the need for three new DupACKs. In this case, NewReno stays in the current fast recovery mode without the need for a new rate reduction step. Another new proposal for improving the fast retransmit/fast recovery algorithm is the Proportional Rate Reduction (PRR) algorithm (Mathis and Cheng 2013).

Even with this fix, NewReno suffers from the fundamental problem because one ACK frame cannot report all the segments lost in a single window. This problem can be solved by using the TCP SACK version (Mathis et al. 1996). TCP SACK can put more acknowledgement information in a single ACK frame because it can utilize the option part of the TCP header. When TCP SACK is used, the receiver uses an ACK format that enables it explicitly to report which segments from a single window made their way to the receiver, and which were lost. With the help of TCP SACK, it is possible to fast-retransmit more than just a single segment per RTT.

In the past ten years, a large number of non-standard TCP variants have been proposed. They often address the under-utilization problem related to the large bandwidth-delay product environments where TCP’s congestion window grows slowly. One such variant is CUBIC TCP (Ha et al. 2008). CUBIC is an implementation in which the window is a cubic function of time since the last congestion event. Being a cubic function, there are two phases for window growth. The first is a concave portion where the window quickly grows to the size of the window before the last congestion event. The next phase is a convex growth phase where CUBIC probes for more bandwidth, slowly at first and then very rapidly. Another major difference between CUBIC and standard TCP versions is that CUBIC's window size does not depend on the number of ACKs received. Consequently, CUBIC allows for more fairness between flows since the window growth is independent of RTT. CUBIC TCP is used by the Linux kernels.

There are also a large number of TCP enhancements. These enhancements try to ensure that TCP really behaves as it should behave in all circumstances. For instance, there are solutions against malicious receivers as well as solutions to make TCP work better over unreliable wireless links. For example, a malicious receiver can acquire more bandwidth by using the ACK splitting technique (also called ACK division attack) (Savage et al. 1999). This is possible because, in its basic mode, the TCP protocol increases its sending rate based on the number of ACKs received. After receiving a data segment carrying 1000 bytes, only one ACK is sent by the honest receiver. This ACK acknowledges all 1000 bytes. In contrast, the malicious receiver can split that ACK. The receiver can generate two ACK messages so that each acknowledges only 500 bytes. This drawback was fixed by Allman (2003), where rate increase is based on byte counting.

3.1.5. TCP fairness

On the Internet, the fairness of a congestion control method traditionally means that a new mechanism is fair regarding the TCP protocol. The reason for that is because most of the flows have been TCP flows and the TCP protocol has traditionally been responsible for the congestion control of the Internet. In relation to today’s Internet, a common definition for Internet fairness is called TCP-friendliness. This kind of TCP-friendly flow is also called a TCP-compatible flow. The basic idea is to protect existing TCP flows from flows that are too aggressive with their congestion control mechanisms. Braden et al. (1998) define this kind of a flow as follows:

A TCP-compatible flow is responsive to congestion notification and in steady state it uses no more bandwidth that a conforming TCP running under comparable conditions.

This definition is somewhat relaxed by Welzl (2005):

A TCP-compatible flow is responsive to congestion notification, and its effect on competing TCP flow is similar to the effect of a TCP flow under comparable conditions.

This second definition makes it possible for a new mechanism to be somewhat better than TCP in some situations. Such a situation could occur when the load level of the network is clearly under the point of the knee. When the network is becoming congested and packets are lost, the new mechanism acts like TCP and, therefore, it is TCP- friendly. On the other hand, when the packet drop rate is at a low level, this new mechanism can be somewhat more aggressive than conventional TCP. The more aggressive behavior is possible if there is no threat of packet losses because TCP primarily reacts only to packet losses.

Unfortunately, TCP fairness is a more complicated matter than these two definitions consider. Even a TCP flow itself is not always friendly toward another TCP flow. There are several different versions of the TCP protocol. These versions cannot be totally identical in their behavior. TCP also has a bias against high-RTT connections because TCP's throughput degrades in the case of higher RTTs (Widmer et al. 2001) as equation 2-7 show. TCP has the property of giving preference to users with short RTTs.

In addition, there are several improvements to make TCP congestion control work in a better way. Only some TCP implementations have adopted these improvements and, therefore, different code implementations behave in different ways. One such improvement is the Delayed ACK mechanism (Braden 1989), which tries to reduce the number of ACK messages. If possible, it acknowledges only every other segment using cumulative acknowledgements. If the congestion window increasing behavior of the sender is based on the number of ACKs instead of bytes, Delayed ACK connections experience a negative bias against dishonest connections that are not using Delayed ACKs.

3.1.6. Explicit Congestion Notification

In their study, Ramakrishnan et al. (2001) describe Explicit Congestion Notification (ECN) for the Internet. ECN is useful because it can reduce packet drops. Notification of the existence of congestion is also reliable. Implementing ECN on the Internet requires more than simply setting a congestion notification bit in the IP header when congestion is experienced. First, the active queue management mechanisms of routers must be activated because it is not worth setting notification bits if packets have to be dropped immediately after congestion marking due to buffer overflow. It is also necessary to react to packet losses because there may still be traffic bursts that are too large to fit into buffers. In addition, using only one control bit for congestion notification is not enough to implement ECN in a proper way. More control bits are needed.

The first two bits reserved for ECN operations have been placed in the Type-of-Service (ToS) field of the IPv4 header or the Differentiated Services (DS) field of the IPv6 header. The first bit is called the Congestion Experienced (CE) bit. It is set when congestion occurs and a packet is marked rather than dropped. The second bit is the ECN Capable Transport (ECT) bit. This ECT bit informs routers that the source is an ECN capable source and will respond to ECN marking as if the packet had been dropped. This bit is required because the Internet can only be upgraded to ECN capable in a gradual manner. There may also be some applications or transport protocols which do not support the ECN mechanism. If the traffic flow is ECN capable, the ECT bit is set by the source and the routers will mark the packet instead of dropping it. If the ECT bit is not set, the packet must always be dropped in the congestion state.

Since this kind of explicit notification is a forward type indication, more control is needed. The indication goes to the receiver and it must be echoed back to the sender. The TCP protocol does this echoing with the help of two extra bits. Both of these bits are placed in the TCP header part, which was originally reserved for future use. The first bit is called the ECN echo bit and it echoes the congestion event back to the sender. This bit is set in the ACK frame, which is sent to the sender. In fact, this bit is set in a series of ACK frames because ACK frames can be sent unreliably. This happens if data is transferred in only one direction and therefore ACKs are not sent in a piggybacking manner. Because an ECN echo can disappear on the ACK path, the sending rate is never reduced because of this congestion event. The problem with unreliable ACKs is solved so that every ACK frame is sent with the ECN echo bit set after the congestion event. This ECN echo setting continues until the receiver receives a data frame where the Congestion Window Reduced bit (CWR) of this data frame has been set. The CWR bit is the other extra bit reserved for the use of ECN. The CWR bit informs the receiver that the sender has received the congestion echo and reduced its sending rate. The CWR bit is transferred reliably because it is sent with a reliable transfer data frame.

More control is necessary if receivers can act maliciously. When a packet is marked, a malicious receiver can easily omit to set the ECN echo bit and the reducing event of the sending rate can be avoided. This problem can be solved with a nonce. The nonce is a random number that is set as part of the data frame when the data frame is sent. In the congested situation, the router must delete the nonce at the same time as the CE bit is set. The idea of the nonce is that the receiver must echo the nonce back to the sender with every ACK frame where the ECN echo bit is not set. If a frame with a set CE bit is received, the receiver must set the ECN echo bit of the next ACK frame because the receiver is unable to echo the deleted nonce. The disadvantage of using the nonce is that more control bits are required in the header. Spring et al. (2003) present a solution where nonce behavior is implemented without the need for extra control bits. There have also been other proposals for implementing ECN, for example, the eXplicit Control Protocol (XCP) (Katabi et al. 2002).

There is also a technology called Pre-Congestion Notification (PCN) (Menth et al. 2013). PCN provides feedback about load conditions in the network to the boundary nodes of the network. PCN uses packet marking to notify boundary nodes if the preconfigured queue threshold has been exceeded on some link. This feedback is used for PCN-based admission control and flow termination. Instead of decreasing the sending rates of connections, PCN uses the feedback to reduce the number of connections in the network. Admission control and flow termination are useful for protecting the quality of inelastic flows. An inelastic flow requires a minimum bit rate. Applications like real-time voice and video are called inelastic because these applications become unusable if they have less than a minimum bit rate at their disposal.

3.2. Delay-based congestion control mechanisms

Delay-based congestion control mechanisms are presented here because delay-based approaches are also utilized in the mechanism developed in this thesis. First, the Packet Pair measurement technique is described. Perhaps the best-known delay-based congestion control mechanism, TCP Vegas, is introduced subsequently. The following techniques, TCP-LP and LEBDAT, are somewhat different from TCP Vegas because they do not try to obtain their own shares of the bandwidth. These two techniques offer low priority services compared to the TCP protocol. A survey of these kinds of low priority transport protocols is presented in a paper by Ros and Welzl (2013). The last technique presented here is the Delay-Gradient Congestion Control Algorithm which uses delay gradients rather than delays for congestion control. Many more delay-based congestion control techniques exist such as DUAL, FAST TCP, and CARD, for example. DUAL (Wang et al. 1992) is one of the seminal works in this area. FAST TCP (Wei et al. 2006) has been called the high-speed version of Vegas and CARD (Jain 1989) is another seminal analytical work in this field.

3.2.1. Packet Pair

In actual fact, Packet Pair (Keshav 1991) is not a congestion control mechanism. It is a measurement technique that can be used to derive the capacity of the bottleneck link. It can be considered as a basic technique for delay-based measurements. In this technique, two packets are sent back-to-back so that the second packet is sent immediately after the first one. Therefore, there is a high chance that these packets are served one after another by the bottleneck link. The delay properties of these two packets can be used to derive the capacity of the bottleneck link.

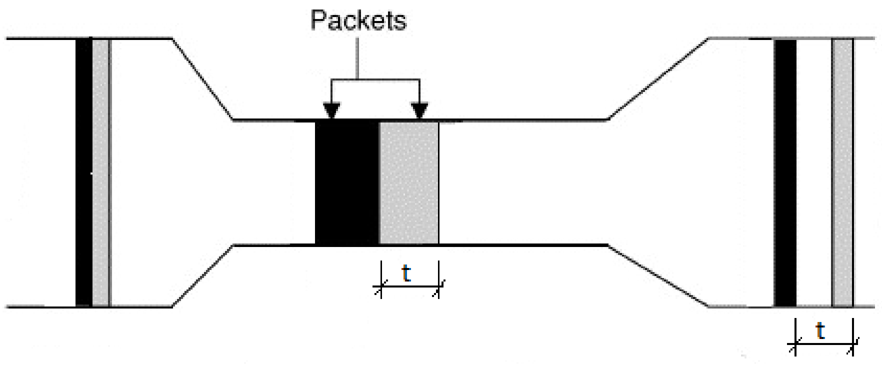


Figure 3.3 Principle of the Packet Pair technique

Figure 3.3 shows what happens when two packets are sent through a bottleneck link. In this figure, there are three links and two packets are moving from left to right. The links on the left and right sides are links with high capacity. The link in the middle is the slow capacity bottleneck link. The shaded areas represent packets. The aim of Packet Pair is that two measuring packets enter the bottleneck link back-to-back and the second packet must be kept in the queue at least for a short time. This guarantees that these packets are sent over the bottleneck link one after another without a gap. Because the number of bits inside the packet is not changed when the packet travels through the bottleneck, the packet must spread out in time. Therefore, these packets leave the bottleneck link with a time gap that depends on the capacity of the bottleneck. If neither of these packets experiences queuing later on the connection path, they will reach the receiver with this time gap. The capacity of the bottleneck link can be calculated by dividing the size of the first packet by the time gap t. This division uses the size of the first packet since the Packet Pair approach was redefined by Lai and Baker (2000) so that a small packet is sent immediately after a large packet. This modification increases the chance of getting these two packets served back-to-back in the bottleneck.

This bottleneck link behavior can also be considered as the background to the self-clocking/ACK-clocking mechanism of the TCP protocol mentioned earlier in this thesis. If the receiver immediately acknowledges incoming packets without delayed ACKs, and if these ACKs do not experience any queuing inside the reverse path, the ACKs are received with spacing based on the behavior of the bottleneck link.

3.2.2. TCP Vegas

TCP Reno repeatedly increases its sending rate in an additive manner. In this way, it tries to find the load level of the congestion and then it backs off significantly away from this level. Consequently, TCP Reno oscillates around the optimal point in a saw-tooth manner. The congestion window size of TCP and the queue lengths of routers also show clear oscillatory behavior. Such behavior is inherent to the AIMD algorithm and this oscillation is used to probe variable network resources.

In contrast, TCP Vegas (Brakmo et al. 1994) employs a different congestion control scheme than TCP Reno. Reno's congestion control works in a reactive manner whereas Vegas' congestion control works in a proactive manner. Instead of using dropped packets as an indication of network congestion, TCP Vegas tries to use network delays as an indication of incipient congestion. Therefore, TCP Vegas adopts congestion avoidance behavior and tries to avoid congestion losses. TCP Vegas also tries to stay at the optimal load level. For this reason, the window evolution of TCP Vegas shows much smoother variations compared with TCP Reno. The strategy of TCP Vegas is to adjust its sending rate (congestion window) in a manner that attempts to keep a small number of packets buffered in the routers of the transmission path.

In the congestion avoidance phase, TCP Vegas estimates the amount of data buffered in the routers. Based on this estimation, the congestion window can be increased, decreased, or left unchanged. Vegas has a variable called $BaseRTT$. Vegas sets this $BaseRTT$ to the minimum of all measured round-trip times of the connection. Typically, this is the RTT of the segment that is sent in the situation when the queues of the routers along the routing path are empty. Therefore, $BaseRTT$ is typically the same as the propagation delay of the round-trip connection path without any queuing delays. With the help of this $BaseRTT$, Vegas computes the expected rate as: $Expected = Cwnd/BaseRTT$, where $Cwnd$ is the size of the current congestion window. Next, Vegas calculates the current actual sending rate. This is done by recording the sending time of the distinguished segment and recording how many bytes are transmitted between the time period when this segment is sent and when its acknowledgement is received. The actual RTT of the distinguished segment is also calculated after receiving the acknowledgement. The actual rate $Actual$ is calculated as the number of bytes transmitted divided by the RTT sample. This calculation is done once per round-trip time.

After calculating $Actual$, Vegas compares $Actual$ to $Expected$ and adjusts the congestion window accordingly. Let $Diff = Expected - Actual$. Note that $Diff$ is always positive or zero by definition since $Actual > Expected$ implies that $BaseRTT$ needs to be changed to the latest sampled RTT. Two thresholds are also defined. If $Diff < low\_threshold$, Vegas increases its congestion window because Vegas concludes that there are too few packets buffered in the routers. If $Diff > high\_threshold$, Vegas decreases its congestion window because Vegas concludes that there are too many packets buffered in the routers. If $Diff$ is between $low\_threshold$ and $high\_threshold$, Vegas leaves the congestion window unchanged.

These low and high thresholds are defined in terms of KB/s. However, it is perhaps more illustrative to think of these thresholds in terms of how many extra bytes the connection is occupying inside the network. Let the size of the queue be denoted by N (bytes). Thus, N can be approximated by: $N = Diff * BaseRTT$. When $N > high\_threshold\_in\_bytes$, Vegas decreases its congestion window to make N smaller.

When $N < low\_threshold\_in\_bytes$, Vegas increases its congestion window to make N larger. We can see that Vegas attempts to keep N between $low\_threshold\_in\_bytes$ and $high\_thresholdRTT\_in\_bytes$.

TCP Vegas also uses a modified slow start algorithm and a modified fast retransmit algorithm. The original TCP slow start requires losses to detect congestion in the network. The modified slow start tries to find the correct window size without generating losses. The basic idea is that TCP Vegas ends its slow start phase and changes its congestion avoidance stage immediately after detecting that a queue is building up on the connection path. This is done by exponentially increasing its sending window only every other RTT. In between, when the congestion window stays fixed, a valid comparison of the expected and actual rates can be made. When there are few packets buffered in the routers, the actual rate falls below the expected rate by a certain amount and Vegas changes from the slow-start mode to its congestion avoidance mode.

TCP Vegas employs a different fast retransmit strategy than TCP Reno by activating retransmissions sooner. Vegas uses a millisecond resolution timestamp for each transmitted segment instead of using a 500 ms coarse-grained timer as Reno does. After receiving an ACK, Vegas retrieves the sent timestamp of the corresponding segment and calculates the RTT in millisecond resolution. Vegas computes the value of the fine-grained retransmission timer just like Reno computes the value of the coarse-grained timer. The fine-grained timer is calculated as the average RTT plus four times its variance. After receiving the first duplicate ACK, Vegas resends the corresponding segment if the fine-grained timer of the segment has expired. In this case, Vegas does not need more than one duplicate ACK to activate the retransmission. When the retransmitted segment is acknowledged, Vegas reduces its congestion window multiplicatively, using a factor of ¾ rather than Reno's ½.

When the next non-duplicate ACK is received after the fast-retransmit operation, Vegas checks whether it is the first or second one after the retransmission. If so, Vegas checks whether the time interval since the segment was sent is larger than the fine-grained retransmission timeout value. If it is, Vegas retransmits that segment. In this way, Vegas tries to solve the problems related to situations in which multiple packet losses occur during one RTT. In TCP Reno, the congestion window may be decreased more than once for losses that occur during one RTT interval. In contrast, Vegas only decreases the congestion window if the retransmitted segment was sent after the last decrease. The packet losses before the last window decrease do not imply that the network is congested because of the current congestion window size, and therefore, do not imply that the current window size should be decreased again.

Unfortunately, TCP Vegas suffers from some problems (Fu 2001). These problems are common for congestion control mechanisms based on delay measurements. The first problem is one of asymmetry. A TCP connection can experience an asymmetry problem if the bandwidth of the data path is much larger than the bandwidth of the acknowledgement path. In such a situation, it may happen that the queues of the acknowledgement path fill up while the queues of the data path remain empty. Therefore, Vegas can incorrectly converge to an operating region in which the available bandwidth of the data path is considerably under-utilized.

Another problem of Vegas is one of rerouting because changing the routing path may change the delay properties of the connection. In a situation like a link failure, the delay properties of the connection usually increase because the optimal path is no longer usable. Because the propagation delay usually increases in such a situation, Vegas maintains the old and too short estimate for the $BaseRTT$. As a result, the data path may be under-utilized. The third problem of Vegas is known as the persistent congestion situation. If a new connection enters the network when queues have already built up, this connection can receive an incorrect assessment of the delay properties of the network. In other words, the value of $BaseRTT$ is overestimated. This can lead to a situation where TCP Vegas keeps more packets in the queues of the routers than it expects. If this situation lasts for a long time, it can lead to persistent congestion.

There is also a TCP version celled TCP Veno (Fu and Liew 2003). This version is based on TCP Reno with some TCP Vegas-like improvements. The basic idea of Veno is that the Vegas improvements help to differentiate between congestive and non-congestive losses. Non-congestive losses can arise, for example, if there are problems with wireless links due to bit errors. When a packet loss is detected by the fast retransmit mechanism, TCP Veno checks if $Diff$ is under $high\_threshold$. If it is, TCP Veno draws the conclusion that this particular packet loss is probably a non-congestion loss due to bit errors. In this case, Veno decreases its congestion window by only 20% instead of halving it like Reno does. If $Diff$ is over $high\_threshold$, the congestion window is halved.

3.2.3. TCP-LP

Kuzmanovic and Knightly (2006) introduced TCP-LP (Low Priority), which is an end-to-end protocol. TCP-LP achieves two-class service prioritization without the support of routers. The new service class offered by TCP-LP is a low-priority service as compared to the normal best-effort service offered by the Internet. If TCP- and UDP-based flows need more bandwidth for their use, and the network capacity is fully utilized, TCP-LP flows withdraw from using the bandwidth. On the other hand, the objective of TCP-LP is to use the excess network bandwidth that is unutilized by non-TCP-LP flows.

The low-priority service is implemented with the help of congestion control. To achieve this low-priority service in the presence of TCP traffic, it is necessary for TCP-LP to detect an incipient congestion earlier than TCP. In principle, the network could provide such an early congestion indicator. For example, TCP-LP flows could use a type-of service bit to indicate that they only need a low-priority service. Based on this indication, routers could use Explicit Congestion Notification (ECN) messages to inform TCP-LP flows only that there is an incipient congestion in the network. However, TCP-LP implements this low-priority service without the need of network support.

The key mechanism of the TCP-LP congestion control is the use of one-way packet delays for early congestion indications. TCP-LP measures one-way packet delays and employs a simple delay threshold-based method for early indication of congestion. This early indication of congestion is deduced by the TCP-LP flow whenever the smoothed one-way delay exceeds the threshold. The threshold resides within the range of the minimum and maximum delay experienced by the connection. With the help of the threshold, TCP-LP can react earlier to congestion than a packet drop-based TCP connection. After the threshold is reached, TCP-LP starts to reduce its sending rate while TCP still continues to increase its sending rate in a normal congestion avoidance manner until a packet drop is experienced. On the other hand, if there is unused bandwidth under the threshold, TCP-LP will adopt it. In order to prevent TCP-LP from overreacting to a burst of congestion indications, TCP-LP ignores successive congestion indications during the next round-trip time if the source has just reacted to a previous congestion indication.

3.2.4. Low Extra Delay Background Transport

Low Extra Delay Background Transport (LEDBAT) (Shalunov et al. 2012) is a congestion control algorithm that possesses experimental RFC status. It is a delay-based algorithm which tries to utilize the available bandwidth on an end-to-end path. It also tries to react to an incipient congestion before actual congestion losses.

LEDBAT is designed for low-priority applications because LEDBAT connections withdraw from using bandwidth after a queuing delay exceeds the predefined target. It is designed for the use of background bulk-transfer applications so that LEDBAT connections do not interfere with the performance of competing flows. LEDBAT can be used as part of transport protocols or applications if the data transmission mechanism can carry timestamps and acknowledgements frequently. At the moment, the main objective is to use it with low-priority application protocols, such as those built on top of the UDP for peer-to-peer (P2P, BitTorrent) applications.

The LEDBAT sender uses one-way delay measurements to estimate the amount of queued data on the data path. When the estimated queuing delay is less than the predetermined target, LEDBAT concludes that the network is not yet congested and increases its sending rate to utilize the free network capacity. When the estimated queuing delay becomes larger than the predetermined target, LEDBAT decreases its sending rate in response to the potential congestion. The sending rate is increased and decreased more aggressively if the queuing delay is far from the target. If no losses are detected, the congestion window is changed according to:

$cwnd(n+1) = cwnd(n) + GAIN * off\_target * bytes\_newly\_acked * MSS \,/\, cwnd(\text{n})$ (3-7)
$off\_target = (TARGET - queuing\_delay) \,/\, TARGET$, (3-8)

where:
$TARGET$= the target delay
$GAIN$= the predetermined increase/decrease factor
MSS = the Maximum Segment Size
$cwnd$ = the congestion window.

Please note that off_target can be positive or negative. If a packet loss is detected, the congestion window is halved in a multiplicative decrease manner.

The paper by Carofiglio et al. (2013) proposes some modifications to the LEDBAT algorithm, i.e., modifying the delay-based decrease term. Instead of using the additive decrease term, it recommends using the multiplicative decrease term, which is continuously driven by the estimated distance from the target. This modification is recommended because they have found that there is unfairness among two competing LEDBAT flows starting at different moments due to the additive decrease component. In addition, the additive increase behavior of LEDBAT leads LEDBAT flows to shorten their rate increase steps towards the target delay value. In consequence, it takes more time to reach the target delay and LEDBAT flows experience reduced efficiency compared to flows using a constant additive increase factor. Since TCP Reno uses the additive increase strategy, the paper proposes that the constant additive increase term also be used by LEDBAT.

3.2.5. Delay Gradient Congestion Control Algorithm

Hayes and Armitage (2011) have proposed a Delay Gradient Congestion Control algorithm (CDG). CDG does not require any knowledge of path-specific minimum RTTs or delay thresholds for congestion notification. CDG seeks the maximum RTT (T_max) seen inside a measuring interval along with the minimum RTT (T_min) seen within the same interval. With the help of these two measured RTTs and one-behind saved history, two gradients are calculated and saved:

$G\_min(n) = T\_min(n) - T\_min(n-1)$ (3-9)
$G\_max(n) = T\_max(n) - T\_max(n-1).$ (3-10)

The maximum and minimum measurements are less noisy than per packet RTT measurements. Nevertheless, CDG uses moving average smoothing for gradients so that the sum of the sample gradients is divided by the number of samples in the moving average window.

At different states of the connection, RTTs and gradients behave as shown in Figure 3.4. When the queue becomes full, T_max stops increasing before T_min stops increasing. The reverse is true for the queue moving from full to empty. Figure 3.4b shows the values of the gradients for these two conditions. Based on these estimates, the state of the queue (full, empty, rising, falling) can be evaluated. Unfortunately, delay gradients are often too sensitive to second order delay fluctuations (Wang et al. 1992). Therefore, CDG does not use delay gradients as the main source of congestion discovery. Delay gradients are used to assist a normal TCP Reno-like congestion control to make the right decisions. For example, gradients are used to differentiate between congestion and non-congestion losses. Only when the queue is full, do packet losses become congestion losses.

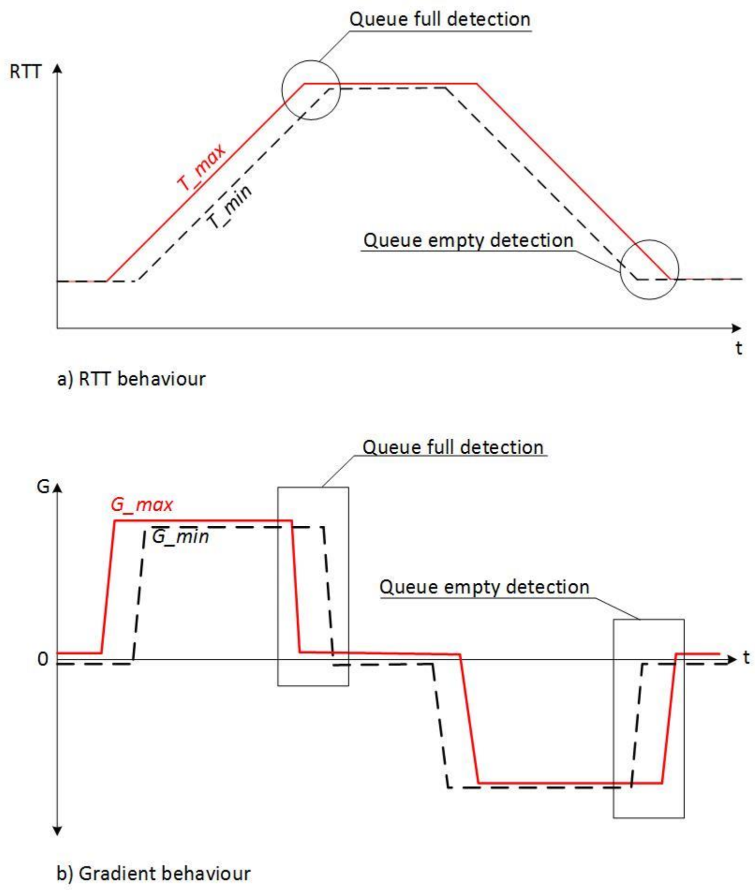


Figure 3.4 Principles of detecting queue states with the help of gradients

3.3. Congestion control for multimedia streaming

In this part of the thesis, some congestion control mechanisms for multimedia streaming are introduced. First, TCP-RTM is described, which suggests some changes to the TCP algorithm. With these changes, TCP would be more suitable for multimedia streaming. After that, some mechanisms are presented, designed to be suitable for multimedia streaming while being TCP-friendly at the same time. The best known this kind of proposal is DCCP and its TFRC version.

Research has been very active in this area during the last few decades and there are many proposals for multimedia streaming which are not introduced in more detail here. For example, Borri et al. (2007) compare window-based and rate-based congestion controls mathematically. They also propose a new rate-based congestion control mechanism called RTP-Primal. Ahmad et al. (2007) introduce a Dynamic Congestion Control Mechanism for Real Time Streams over RTP (DCCM). DCCM uses packet inter-arrival jitters to detect transient congestions and estimated packet loss rate probabilities to detect permanent congestions. The jitter-based approach is interesting. If the bottleneck link is shared between many connections, the jitter actually increases as the queue fills up. There has also been research work to accommodate the SCTP protocol (Stream Control Transmission Protocol) for the use of multimedia streaming (Kim et al. 2010).

3.3.1. TCP for real-time multimedia applications

Liang and Cheriton (2002) investigated the kinds of changes needed for TCP and its use to make it more suitable for real-time applications. They found that after relatively modest extensions to TCP and several application techniques, real-time applications can operate well over TCP. Based on these findings, they developed a TCP version called TCP-RTM (TCP for real-time mode). TCP-RTM is not a completely new protocol but rather an extension of TCP. It can be switched on by using an appropriate socket option. In this way, all TCP’s services, including error recovery, are available to real-time applications. Real-time applications may benefit from retransmissions. The content of real-time media may be significantly compressed to reduce data rate requirements and, therefore, the data stream becomes sensitive to losses. Retransmissions are also sometimes possible because these applications often use significant playback buffers in the order of hundreds of milliseconds to hide the network jitter from the application. In addition, it can be much easier to deploy TCP-RTM than a completely new transport protocol. Using TCP is encouraged by firewalls, which may block some non-TCP connections.

Below is a list of the main TCP-level modifications and application-level techniques that are required to allow real-time applications to use TCP:

- The head-of-the-line blocking problem is solved in the following manner. If the received date is not in the correct order for reading, but one or more out-of-order packets are queued for the connection, the first contiguous range of out-of-order packets is moved from the out-of-order queue to the receive queue. This moved data is delivered to the application. If the skipped data portion arrives later, it is discarded.

- If the playback buffers are large enough, it is possible that quickly retransmitted segments can reach the destination before the play-out moment. Therefore, the length of the playback buffer should be at least $3/2 * RTT + 3 * period + delta$, where $period$ is the typical interval between the data segments and $delta$ is some modest extra time. This formula is based on the assumption that if one segment is lost, the sender receives the third duplicate ACK after $RTT + 3*period$.
- TCP-RTM applications use application-level framing to recognize TCP-level packet losses. TCP-RTM also works in a framing mode instead of working in a normal byte-stream mode. When this mode is enabled, TCP provides framing support so that application-level frame boundaries always align with TCP segment boundaries.
- The sender of TCP-RTM keeps track of the per-connection loss counter that counts the number of segments that have failed to reach the receiver. These packets have been lost in the network and skipped over by the receiver. The number of receiver-skipped-over packets is reported to the sender by the acknowledgement option. The value of the loss counter can be used by the application to adjust its encoding scheme and transmission rate to be suitable for the network condition.

In spite of many improvements, there are still TCP rate fluctuations present with this solution because nothing has been done to smooth the sending rate.

3.3.2. Rate Adaptation Protocol

TCP is not a perfect protocol for all streaming media because it uses retransmissions with head-of-the-line blocking for reliable communication. Rejaie et al. (1999) stated that the easiest way to implement a congestion control oriented, TCP-friendly transport layer protocol for multimedia streaming is to remove TCP’s reliability. They introduced the Rate Adaptation Protocol (RAP), which uses this approach for implementing TCP-friendly congestion control. RAP is one of the first congestion-oriented transport protocols for multimedia applications and it employs an AIMD congestion control algorithm to imitate TCP. Its primary goal is to be fair and, in particular, TCP-friendly while separating network congestion control from application-level reliability.

The RAP protocol is mainly implemented by the sender. The RAP sender sends data packets with sequence numbers and the RAP receiver acknowledges each packet to provide end-to-end feedback. Each acknowledgment packet contains the sequence number of the corresponding delivered data packet. The RAP source can detect losses and sample RTTs by using this feedback. If no losses are detected, the sending rate is increased in an additive manner. RAP searches for the available bandwidth on the bottleneck link by periodically increasing its transmission rate in a similar way to TCP's congestion avoidance phase. It generates losses as TCP does. RAP considers losses to be congestion signals and uses two mechanisms to detect losses: the ACK-based and timeout-based mechanisms. The ACK-based loss detection mechanism is based on the same intuition as the fast retransmit mechanism of TCP. If the RAP source receives an acknowledgement that implies the delivery of three packets after a missing one, the packet is considered to be lost. In contrast, timeout-based packet losses are detected with the help of a Transmission History Table. For every packet, its sequence number and departure time are saved in this table at the same time as the packet is sent. Before sending a new packet, the source checks for potential timeouts among the packets in this history. Timeout values are based on the smoothed RTT estimate.

RAP is also more suitable for multimedia than TCP because RAP's congestion control is rate-based. As stated above, multimedia sources are usually more willing to use rate-based control than window-based control. The transmission rate of RAP is controlled by adjusting the inter-packet gap. For example, after detecting congestion, the transmission rate is decreased multiplicatively by increasing the value of inter-packet gap. RAP does not do much to smooth the allowed sending rate. Therefore, RAP is suitable only for real-time adaptive multimedia data transmission with intelligent buffer control.

3.3.3. Enhanced loss-delay based adaptation algorithm

The enhanced loss-delay based adaptation algorithm (LDA+) (Sisalem and Wolisz 2000) is a rate-based congestion control mechanism that utilizes many earlier developed techniques. LDA+ adapts the transmission rate of UDP-based multimedia flows in a TCP-friendly manner. LDA+ does this by emulating AIMD behavior. It uses the RTCP protocol as a feedback channel to obtain information about packet losses and round-trip times. This means that there is usually much less feedback information than with mechanisms where the receiver acknowledges almost every packet. The amount of RTCP traffic is based on the amount of data traffic so that RTCP makes up a certain percentage of the data rate (usually 5%) with a minimum interval of five seconds between two RTCP messages. RTCP messages are suitable for feedback information because those messages already include information for loss detection and RTT calculation. Because of this RTCP feedback channel, the natural place of the LDA+ algorithm is to be part of the RTP protocol.

LDA+ adjusts its increase and decrease factors on the fly. For this adjustment, LDA+ estimates the bottleneck bandwidth of the connection through the Packet Pair approach.
It is possible to add application-defined extension parts to the RTCP header. LDA+ adds an application-defined part, which includes the sequence number of the data packet that

will start the stream of probe packets and the number of probe packets that will be sent. After that, the data packets starting from this sequence number are sent using the access speed of the end system. The maximum access rate is used to get the probe packets queued one after another at the bottleneck. On the receiver side, the bottleneck capacity is calculated by dividing the size of the probe packet by the gap between two probe packets.

Based on this loss-delay and bottleneck capacity information, the sender adjusts its sending rate. If a packet loss is detected based on the RTCP feedback message, the case is interpreted as a loss situation. In the loss situation, LDA+ decreases its rate so that:

i. The current rate is multiplied by a parameter value less than one.
ii. The TCP throughput equation (equation 2-7) is used to calculate the new sending rate.

A minimum value is chosen for the new rate. If packet losses are not detected, the sending rate is increased. The additive increase step used is not constant. It depends on the share of the bottleneck capacity so that connections that are using slow sending rates become bigger in additive steps. Because LDA+ is used by multimedia streaming, it tries to smooth rate changes.

LDA+ has both advantages and disadvantages (Widmer et al. 2001). LDA+ adapts its rate increase behavior so that it can prevent overshooting the bottleneck bandwidth. Unfortunately, RTCP reports are generated infrequently. This makes LDA+ react slowly to changes in the network condition. RTCP's capability to report small loss rates is also limited. In environments with very low loss rates, RTCP reports zero losses and LDA+ may claim more than its fair share.

3.3.4. TEAR: TCP Emulation At Receivers

TEAR (Rhee et al. 2000) emulates the behavior of the TCP protocol but shifts most of the control functions to the receiver side. TEAR also implements smoothing operations because large rate fluctuations are undesirable with multimedia streaming. After smoothing, the sender is informed about the emulated rate so that the sending rate can be updated. This feedback can be sent much less frequently than in the case of the TCP protocol. For this reason, TEAR is suitable for multicast and asymmetric networks.

TEAR uses the same signals as TCP to determine congestion situations. Such signals are packet arrivals, packet losses, and timeouts. Using these signals, the TEAR receiver emulates the congestion control functions of the TCP sender including the slow start, fast recovery, and congestion avoidance algorithms. For example, receiving three packets after the lost packet triggers the three duplicate acknowledgment behavior and fast recovery is activated. TEAR also maintains the congestion window and slow start threshold variables like TCP does. With the help of this kind of TCP emulation, the receiver can estimate the TCP-friendly rate for congestion control purposes.

TEAR maintains a congestion window variable, *cwnd*, on the receiver side and updates this window according to the same rules as TCP. In theory, the allowed occasional sending rate can be calculated by dividing *cwnd* by RTT. In fact, TEAR does not measure RTTs. In TEAR, a transmission session is partitioned into non-overlapping time periods called rounds. A new round begins when the current round ends. The round contains the arrival of the *cwnd* number of packets. At the beginning of the round, the receiver has the *cwnd* at a certain value. The receiver measures how long it takes to receive this *cwnd* number of packets. This time is now used as an RTT sample. At the end of each round the values of *cwnd* and RTT are saved. Next, the rounds are grouped into longer time units called epochs. Simply speaking, the epoch ends as a packet loss is detected and the next epoch starts at the same time. At the end of each epoch, the receiver divides the sum of all *cwnd* samples recorded in that epoch by the sum of the RTTs in that epoch. This result is called the rate sample of that epoch and is saved in the epoch table. To filter out the noise and smooth the sending rate, the weighted average is calculated over the last eight samples of the epoch table. The weighted average is sent to the sender to control the sending rate.

Rhee et al. (2000) indicate that TEAR shows superior fairness to TCP with significantly lower rate fluctuations than TCP. TEAR’s sensitivity to the feedback interval is very low. Therefore, even under high feedback latency, TEAR flows exhibit acceptable performance in terms of fairness, TCP friendliness, and rate fluctuations.

3.3.5. Datagram Congestion Control Protocol

Since UDP does not offer congestion control, it is the responsibility of applications to implement appropriate congestion control functions. As it is not easy to program TCP-friendly congestion control into an application, a protocol called the Datagram Congestion Control Protocol (DCCP) was developed by IETF. The basic idea is that this protocol is part of the operating system and it offers congestion control for unreliable applications. DCCP can briefly be described as TCP without byte-stream semantics and reliability, or like UDP with added congestion control, handshakes, and acknowledgments for congestion feedbacks.

Congestion control is not part of DCCP itself. Instead, DCCP allows applications to choose from a set of congestion control mechanisms. Different kinds of congestion control mechanisms can be used with DCCP. Therefore, at least in theory, the congestion control mechanisms developed in this thesis can be used as part of the DCCP world. At this moment, two main mechanisms have been specified: TCP-like Congestion Control (Kohler and Floyd 2006) and TCP-Friendly Rate Control (TFRC) (Floyd et al. 2008). The

TCP-like mechanism probes the available bandwidth aggressively. It is suitable for applications that want to use as much bandwidth as possible but can tolerate significant rate fluctuations. TFRC is designed for applications that prefer a smooth rate. Many real-time multimedia applications are such applications. Therefore, TFRC is discussed in more detail later after a brief introduction of the main DCCP protocol.

3.3.5.1 Main DCCP protocol

Most of the congestion control mechanisms described in this thesis are merely schemes and not complete protocols. A transport protocol often includes many functions that are not necessarily congestion control specific. For example, there are checksums for error checking and port numbers to ensure that applications can be identified. DCCP offers all the necessary functions. Kohler et al. (2006b) introduce the core DCCP protocol. The paper by Kohler et al. (2006a) is also significant because it discusses the motivations behind the design of DCCP. Their paper explains the functions needed for this kind of transport protocol.

DCCP is a connection-based protocol even though it is an unreliable protocol. This guarantees better cooperation with middle-boxes, like firewalls. It also creates ground for feature negotiation. One important feature is CCID, Congestion Control Identity. Every approved congestion control mechanism has its own number for CCID. DCCP connections are bidirectional. Data may pass from both endpoints to the other or only in one direction. Acknowledgements can also be piggybacked by data packets. Logically, however, a DCCP connection consists of two separate unidirectional connections called half-connections. It is possible that a half-connection uses its own congestion control mechanism, which is different to that used by the other half-connection.

3.3.5.2 TCP Friendly Rate Control

TCP Friendly Rate Control (TFRC) (Floyd et al. 2008) is designed to be reasonably fair when competing for bandwidth with TCP flows. It is difficult to define TCP friendliness exactly but Floyd et al. (2008) state that TFRC is "reasonably fair" if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions. TFRC has a much lower variation in sending rate over time compared to TCP. This smoothing behavior makes it suitable for applications such as streaming media where a relatively smooth sending rate is important. With a smoothed sending rate, TFRC responds to changes in available bandwidth more slowly than TCP. For this reason, TFRC is not recommended for applications that simply need to transfer as much data as possible in as short a time as possible.

TFRC is best suited for applications that use a fixed packet size and respond to congestion by varying the sending rate in packets per second. TFRC can also be used with applications that do not have a fixed packet size, but where the packet size varies according to the needs of the application (for example with video applications). With these kinds of applications, TFRC perhaps does not work in the optimal way. TFRC is not suitable for applications that require a fixed interval of time between consecutive packets and vary their packet size in response to congestion. TFRC is more a receiver-based mechanism than a sender-based mechanism. Therefore, TRFC is well-suited for an application where the sender is a multi-connection server and the receiver has more processing power for congestion control computation.

TFRC is an equation-based congestion control mechanism. It uses a throughput equation to calculate the allowed sending rate. Because DCCP tries to be fair toward TCP, TFRC uses the TCP throughput equation. This equation describes the TCP sending rate as a function of the loss event rate, round-trip time, and segment size:

$$B(p) = \frac{s}{RTT\sqrt{\frac{2p}{3}} + \left(4RTT * 3\sqrt{\frac{3p}{8}}\right)p(1+32p^2)}\ , \qquad (3-11)$$

where:
$B(p)$ = the sending rate bytes/sec
RTT = the round-trip time
s = the segment size in bytes
p = the loss event rate.

Equation 3-11 is the same equation as 2-7 but with the TCP retransmission timeout value set to the value of 4*RTT and the sending rate presented in bytes. It is also supposed that the maximum number of packets acknowledged by a single TCP acknowledgement is one. On the other hand, the unanswered question is why the factor of 1.3 is replaced by the factor of 3.

To simplify, a loss event interval involves a group of packets between two consecutive packet losses. The duration of the loss event interval is at least one RTT so that TFRC does not react to multiple packet losses inside the same window. The loss event rate is the inverse of the number of packets in the particular loss event interval. The receiver measures the weighted average loss event rate. The receiver sends this average information back to the sender. The sender also uses these feedback messages to measure RTTs. The loss event rate and smoothed RTT estimate are then fed into TFRC's throughput equation to obtain the allowed sending rate, which is limited to be at most twice as high as the receive rate. To obtain this receive rate, the receiver continually calculates the received data rate and reports it back to the sender.

TFRC is a well-designed congestion control mechanism. All kinds of exceptions have been taken into consideration. For example, if the sender does not receive any feedback report during four round-trip times, the sender halves its sending rate. This is accomplished by a timer known as the no-feedback timer. The so-called data limited period has also been taken into consideration. The data limited period means that the sender cannot send data using the allowed rate because the application layer is unable to deliver enough data to the DCCP entity. There is also an initial slow-start phase, which continues until the first loss event is experienced. In addition to being a well-designed mechanism, TRFC has also been well analyzed by many research studies. These studies often suggest some improvements. For example, Li and Chen (2005) suggested that jitters could be used as a warning for congestion.

In fact, TFRC also contains a delay-based congestion control property, although this property is an optional part of the implementation, which is used to reduce oscillation. The sender maintains a long-term estimate of the square root of RTT. The sender modifies its sending rate depending on how the square root of the RTT sample differs from the long-term estimate. When the square root of RTT is greater than the long-term estimate, it implies that the queuing delay is probably increasing. In this case, the sending rate is decreased beforehand to minimize oscillation.

There is also the Small-Packet variant of TFRC (TFRC-SP) (Floyd and Kohler 2007) that has been designed for applications that send small packets. It uses the same TCP throughput equation as TFRC. Instead of using an actual segment size in the equation, TFRC-SP version uses a nominal segment size. A typical value for the nominal size is 1460 bytes. In addition, the loss event rate is calculated in a slightly different way. Packet headers have also been taken into account. The allowed sending rate presented in bytes per second is reduced by a factor that takes into account the header sizes of both the network and transport layers. TFRC-SP enforces the use of a minimum interval of 10 milliseconds between data packets to prevent a single flow from sending small packets arbitrarily often.

3.3.6. TCP Friendly Window-based Congestion control

TFRC and its friendliness toward TCP have been investigated in many research studies such as Rhee and Xu (2007), for example. It has been found that there is a remarkable imbalance between TFRC and TCP sources in some conditions. For example, under a low level of statistical multiplexing where only a couple of flows coexist, TCP can have a higher loss event rate than TFRC. To solve this problem, Choi and Handley (2009) propose a window-based version of TCP Friendly Rate Control. This window-based version is called TCP Friendly Window-based Congestion control (TFWC).

TFWC senders use the TCP throughput equation to compute congestion windows. The equation for TFWC can be derived from the simplified TCP throughput equation (Eq. 3-11). In equation 3-11, s is the packet size in bytes and RTT is the end-to-end round-trip time. In window-based congestion control, it is normally supposed that it takes one RTT to send the whole window. Thus, equation 3-11 must be multiplied by RTT to obtain the equation for the window-based mechanism. In addition, if this equation is divided by s, we obtain the window in packets. If both sides are multiplied by RTT/s, we get the congestion window size W in packets:

$$W = \frac{1}{\sqrt{\frac{2p}{3}} + \left(12\sqrt{\frac{3p}{8}}\right)p(1 + 32p^2)} \qquad . \qquad (3-12)$$

This equation is interesting because it shows that only the loss event rate p has to be known in order to calculate the TCP-friendly window size.

To calculate the loss event rate, TFWC uses the same average loss interval mechanism as TFRC. In this mechanism, the last eight intervals between the packet losses are stored and the weighted average is calculated. In TFWC, the loss event rate is calculated by the sender. For feedback information about packet losses, the receiver sends ACK messages to the sender. The ACK message carries an Ack Vector where the delivery status of each packet is reported. For the ACK messages, the receiver maintains an Ack Vector data structure that contains the packet list indicating whether or not a packet was delivered successfully. The Ack Vector grows in size when the receiver gets packets from the sender. Therefore, the ACK message itself needs to be acknowledged so that already acknowledged packets can be removed from the Ack Vector.

As equation 3-12 shows, RTT measurements are not required with this kind of congestion control mechanism. It leads, however, to very bursty behavior. In window-based control, the idea is that the sender sends the whole allowed window as soon as possible. When the window closes, the sender must stop and wait until the window opens again. This bursty behavior is somewhat alleviated because TFWC switches back to the TFRC mode (rate-based but still sender-based) when the window is small. In fact, TFWC also implements RTT measurements. RTTs are needed to calculate the retransmission timeouts. With the help of RTO timers, deadlock cases can be solved. A deadlock can happen if the Ack clock stalls due to packet drops. Because TFWC is an unreliable protocol, it actually sends a new packet rather than a retransmission after the timer expires. The window size can be divided by RTT to remove the bursty behavior. However, we are then again in rate mode.

There are also other TFRC variants such as MulTFRC (Damjanovic and Welzl 2011) and Enhanced TCP-Friendly Rate Control (ETFRC) (Talaat et al. 2013), for example.

MulTFRC is a TFRC extension that enables its users to multiplex several application-level streams into one TCP flow. This composite TCP flow can receive the same bandwidth as it would have gotten if the application had actually used separate TCP flows. ETFRC modifies TFRC's algorithm so that its performance surpasses TFRC in terms of throughput, jitter, and packet loss.

3.3.7. Google Congestion Control for Real-Time Communication on the World Wide Web

The paper by Carlucci et al. (2016) describes methods for congestion control when using real-time communication on the World Wide Web (WebRTC). This WebRTC means that all control functions are implemented inside the web browser on the application layer, perhaps with the help of the RTP protocol. The document defines two congestion control methods, one for the sender side and another for the receiver side.

The receiver side method is based on one-way network delays. Each video frame is equipped with a timestamp $T(i)$ when transmitted. Here, i is the sequence number of the video frame. After receiving this frame, the arrival time $t(i)$ is also defined. If this frame is delayed/queued, we obtain:

$$t(i) - t(i-1) > T(i) - T(i-1). \tag{3-13}$$

This means that the arrival time difference is larger than the timestamp difference. The relative inter-arrival time difference can now be defined:

$$d(i) = t(i) - t(i-1) - \big(T(i) - T(i-1)\big). \tag{3-14}$$

After each frame received, these relative inter-arrival time differences are fed to a sophisticated over-use detector. The aim of the over-use detector is to minimize the queuing delays in the buffers along the connection path. It estimates whether the connection path is underused, overused, or in normal state. Based on the observed state, the sending rate is increased or decreased in a multiplicative manner, or it is kept unchanged.

The sender side method is based on information regarding packet losses. This information is delivered to the sender by means of RTCP messages. Based on the fraction of lost packets, the sending rate is adjusted according to the MIMD approach. If a high fraction lost is observed, the rate is multiplicatively decreased. When the fraction lost is considered negligible, the rate is multiplicatively increased. The allowed sending rate is the minimum of the sending rates produced by both sides. Google has also introduced another new congestion control algorithm, which Google calls BBR, standing for Bottleneck Bandwidth and Round-trip propagation time (Cardwell et al. 2016).

3.4. TCP/IP Video Transfer

This section introduces issues related to current Internet-based video transfer. Different video streaming modes are introduced. Three main categories for implementing video transfer are UDP streaming, progressive download, and HTTP adaptive streaming. Although all three types of approaches are used in practice, adaptive HTTP streaming is employed by the majority of today’s major video streaming providers, such as Netflix and YouTube. Problems with current practices are highlighted at the end of this section to justify the need for better congestion control practices for video services.

3.4.1. Quality metrics

Transferring video streaming over the best effort based Internet renders data transfer vulnerable to different kinds of impairment. Such as packet losses, delays, and jitters can degrade the quality of the videos as experienced by the end users. The problem of defining video quality and user experience has produced a variety of measuring methods for video quality. QoS parameters have traditionally been objective metrics that define the performance of IP-based services. QoS metrics use easily measurable network parameters to define video quality, such as connection speed, transmission delay, traffic jitter, and packet loss rate. QoS parameters are highly network-oriented and QoS metrics cannot fully quantify end users’ perception of quality. Therefore, video quality measurement is currently migrating towards measuring Quality of Experience (QoE) (Staelens et al. 2015). One reason behind this change is also the widespread use of HTTP adaptive streaming techniques.

QoE (Seufert et al. 2015) emphasizes the end user's subjective assessment of video quality. QoE consists of many different types of factors whose mutual importance is also influenced by the end user's personal expectations. QoE can be evaluated by using subjective and objective methods. Subjective evaluation consists of measuring real users’ QoE opinions. Objective evaluation includes measurable network dependent technical parameters, such as video initial delay, bit-rate, stalling, re-buffering, and bit-rate switching. Stalling refers to the stopping of video playback due to insufficient data in the playback buffer. Stalling can appear if there is not enough bandwidth for the video download or there are some shortcomings in the operation of the rate adaptation techniques. The playback is interrupted until the buffer reaches a sufficient level again. Re-buffering is synonymous with stalling because this buffer filling stage during stalling is called re-buffering. Dobrian et al. (2013) have found that re-buffering has a major impact on QoE.

Many QoE models have been presented for measuring and evaluating perceived video quality (Juluri et al. 2016). Since the effects of QoE are partially caused by the QoS problems of the network, there are also many QoS/QoE correlation models (Hsu et al. 2013; Kim et al. 2012) where QoS parameter measurements can be used for QoE evaluation.

3.4.2. Content Distribution Networks

All major video streaming companies utilize Content Distribution Networks (CDN), also known as Content Delivery Networks, to distribute videos to their customers. A CDN uses a massively distributed server infrastructure to cache content near the customers. It manages server clusters in multiple geographically distributed locations and attempts to direct a user request to a CDN location that offers the best QoE. Utilizing CDN in video delivery provides many benefits. It offers reduced delivery cost for cacheable content. It reduces latency and, consequently, improves QoE. The distributed service also improves the robustness of delivery because a single point failure cannot crash the whole system. Furthermore, the threat of congestion is also diminished.

There are two different primary strategies for replicating content across a CDN. In the push-based approach, content is distributed proactively to edge servers, known as surrogate servers. When the end user sends a request for a video, the video material is already ready for download on the surrogate server. The primary server (called the back-end server) pushes videos to the network after the video material to be shared has been created or updated. For example, the Netflix CDN (Böttger et al. 2016) uses the push-based strategy where content is pushed to the surrogate servers during off-peak hours. In the pull-based approach, the video material is not cached on the server until the first end user asks for the video. When the first end user asks for a specific video, the surrogate server has to pull the video material from the primary server at the same time as it sends the video to the end user. Once the video has been pulled from the primary server, it is cached and served directly from the surrogate server until the cached video material expires. The pull-based strategy minimizes storage space. The YouTube CDN (Hoßfeld et al. 2013) operates in a pull-based approach. In these strategies, the data transfer used can be based on the FTP protocol or HTTP-based approaches, for example.

An important component of CDN architecture is a cluster selection strategy, also called a request routing mechanism. This dynamically directs users’ requests for content to the appropriate server based on a specified set of rules. For example, a simple and widely used strategy is to direct a video request to the geographically closest cluster where the content is cached. However, sophisticated strategies use more complex rules for balancing the load between servers (Hoßfeld et al. 2013). In some systems, the serving server can be changed to another during the video streaming session to balance the load or to improve QoE.

Load balancing and request rerouting can be implemented in DNS-based, application layer based balancing, or transport layer based approaches (Manfredi et al. 2013). With a DNS-based approach, a specialized DNS server is able to execute server selection. For every address resolution request received, the DNS server selects the most appropriate server and provides the client with the IP address of the selected server. In the application layer based approach, the primary server application software selects the appropriate server. In the case of the Web server, the requested Web server can decide either to serve the request or redirect the request to another server. An HTTP redirect message can be used to inform the client of the address of the other server. In the less commonly used transport layer based approach, the header structures of the request message are analyzed in order to select the most appropriate server. The client's IP address is the most important information concerning the request message. Rerouting to the selected server can be achieved either by rewriting the IP destination address of each incoming packet, or by a packet-tunneling mechanism. CDNs often have separate servers for control functions and video streaming functions.

3.4.3. Classical video streaming

In UDP streaming, a multimedia data stream is typically encapsulated inside RTP messages, or inside equivalent packet formats. These RTP messages are transferred over UDP, even if TCP can also be used. Since UDP does not implement flow control and congestion control, the media stream can operate without any inherent rate control mechanisms. This allows the server to push packets to the client at a bit-rate that depends on application-level implementation issues rather than the underlying transport protocol network issues, and makes UDP streaming suitable for low-latency and best-effort media transmission (Begen et al. 2011). In this kind of push-based streaming, the server transmits content roughly at the media encoding bit-rate, which also matches the client's media consumption rate. Therefore, UDP streaming can use a small receiver buffer size. This promotes the efficient use of bandwidth in video interrupt cases because only a part of the video that is sure to be watched is passed over the network.

However, UDP streaming has its shortcomings (Ma et al. 2011). UDP streaming requires a specialized server that implements application-level algorithms for control tasks, such as bit-rate and retransmission management. Because a media stream is sent in a continuous manner, the server has to maintain a session with the client to listen to commands from the client for session-state changes. This requires resources from the server if there are large numbers of serving clients. If the bandwidth drops below the encoding rate, the receive buffer can underflow, which interrupts the playback. The risk of stalling can be alleviated by using multi-layer codecs (Seufert et al. 2015). Another drawback is that firewalls may be configured to block UDP traffic.

In progressive download, also called HTTP streaming, the client requests a video using HTTP request messages. The server delivers the video file to the client over the TCP connection using the HTTP response messages. In its basic form, progressive download is not a genuine streaming technique because it creates a temporary copy of the video file on the client device. It looks like streaming since video playback can be started once the initial part of the video has been loaded onto the device. The entire video can be downloaded with a single HTTP request. However, progressive download is often implemented using HTTP range GETs to request the video file in segments in a paced manner (Ma et al. 2011).

The main advantage of progressive download is that there is no need for dedicated server software. A normal web server can also act as a video server. The use of HTTP over TCP also eliminates firewall blocking. Progressive download also has its disadvantages. Compared to HTTP over TCP, UDP streaming imposes a lower transmission overhead. Due to the use of TCP protocol on the transport layer, it does not allow the use of the genuine multicast mode for video distribution. Packets can be delayed significantly due to the TCP's retransmission mechanisms. Since the sending rate used is controlled by TCP congestion control, the transmission rate can fluctuate significantly. The video can stall and re-buffering is needed for viewing to continue if the bandwidth is under the video encoding rate for a long time. Some of these shortcomings can be alleviated by using server-side or client-side bandwidth management techniques for coarse-level rate adaptation but HTTP adaptive streaming techniques offer a more modern solution.

3.4.4. HTTP adaptive streaming

The best-effort basis Internet cannot provide guaranteed bandwidth for a video stream. If a fluctuating bandwidth is not sufficient for a constant bit-rate downloaded video for a given period of time, the client side application starts to consume the buffered video at a greater rate than that at which new data is being received. If the buffer eventually runs out of data, the video stalls and re-buffering is needed. This phenomenon can be avoided if the quality of the video presentation is adapted to current network conditions by using HTTP Adaptive Streaming (HAS), or its standardized version Dynamic Adaptive Streaming over HTTP (DASH) (Sodagar 2011). The main goal of DASH is to prevent client's playout buffer under-run while maximizing the QoE of the video user by adapting the video rate to the fluctuating network conditions. As rate adaptation decisions are made mainly on the client side, the rate adaptation can also take into account the features of user devices.

DASH encodes video content into multiple bit-rate versions. Each encoded video version is fragmented into small video segments or chunks, each containing typically from 2 to 15 seconds of video. In DASH, the client can smoothly switch between different bit-rate versions at the chunk boundaries because chunks are timed in the video time line. The video quality and bit-rate selections are based on the network throughput, the current receive buffer status, and the Media Presentation Description (MPD) file. MPD, also called a manifest file, provides information about the chunks, such as available bit-rates, codecs, byte ranges, and download URLs. Video chunks and MPDs are fetched from the server using standard HTTP procedures so that the client manages the streaming without having to maintain an application-level session state on the server.

DASH is not an actual congestion control mechanism. Unlike actual congestion control mechanisms, DASH does not control the video transmission rate directly. Since DASH is layered on top of TCP, the current DASH sending rate is based on TCP's dynamic rate adaptation behavior, In DASH, the appropriate bit-rate selection is based on two control loops (Kua et al. 2017):

- The TCP congestion control loop reacts to network congestion and tries to select the sending rate that is suitable for the connection path.
- The DASH loop reacts to the sending rate that TCP dictates and tries to adapt the bit-rate to the average observed TCP throughput.

The DASH control loop is too tardy for the use of congestion control. The TCP control loop typically reacts over a time period that is well below one second, whereas the DASH control loop performs adaptation over a time period of several seconds. DASH tries to smooth the sending rate, ignoring the short-term rate fluctuation of TCP.

DASH implementations are not compatible with each other. The MPEG-DASH standard does not enforce any particular adaptation behavior; it merely defines the MPD and segment formats. The delivery of the MPD, the media-encoding formats, the client adaptation behavior, and the video-playing techniques are outside of MPEG-DASH's scope. Some major companies also have their own solutions, which do not comply with the MPEG-DASH standard. MPEG-DASH does not even mandate the use of any specific transport layer protocol, allowing flexibility for emerging transport layer protocols.

There are many proposals for the rate adaption of DASH (Kua et al. 2017). Most of the proposals are related to bit-rate selection on the client side, due to ease of implementation and scalability. Client-side adaptation techniques can be classified into buffer-based, throughput-based, and hybrid-based solutions. In buffer-based solutions, the bit-rate is selected directly as a function of the current receive buffer level. Rate selection is more conservative when the buffer threatens to deflate, and more aggressive when the buffer is close to full or above the specified threshold. In throughput-based approaches, the application layer tries to estimate the TCP throughput for the feedback signal of the rate selection. This kind of algorithm typically derives the TCP throughput from comparing the chunk fetch time with the media playback content in the chunk. After selecting the appropriate rate for the next chunk, the algorithm checks the current buffer occupancy level. If the buffer threatens to overflow, the algorithm will delay the request for the next chunk. This kind of adaptation typically generates an alternating on-off traffic pattern.

The player is in the on state when it is downloading a chunk, and otherwise in the off state. Hybrid-based adaptation algorithms take into account both the buffer occupancy level and the TCP throughput when selecting the rate for the next chunk. In this way, they typically alleviate the on-off traffic pattern. ELASTIC (De Cicco et al. 2013) is an example of this kind of hybrid approach. There are also several server-side, network-level, and transport layer solutions for implementing, assisting, or optimizing DASH-based rate selection (Kua et al. 2017).

However, major video streaming companies do not usually reveal which kinds of adaptation techniques they use in their DASH implementations. For example, YouTube adopted DASH as its default playback mechanism at the beginning of 2015. Implementation solutions for the YouTube algorithm can be deduced only through various experimental methods. For example, Mondal et al. (2017) explored YouTube’s implementation of the DASH client. They identified important parameters in YouTube’s rate adaptation algorithm and the behavior of the algorithm by analyzing the contents of HTTP request and response message headers. They observed that the video quality adaptation of YouTube’s DASH is based on buffer size distribution at the YouTube client. In addition, they found that YouTube makes an effort to adaptively decrease the lengths of the downloaded video chunks before downgrading video quality.

3.4.5. Problems due to TCP congestion control

This section focuses on the problems of DASH-based video streaming because DASH-based solutions represent the vast majority of current practices. However, TCP's congestion management produces similar problems for progressive download. The problem with UDP streaming is that it does not implement congestion control at all. Therefore, greedy UDP streaming connections can capture the capacity share of other connections.

Wang et al. (2008a) observed that TCP generally provides good streaming performance when the achievable TCP throughput is roughly twice the video bit-rate leading to the waste of half of the bandwidth. They found inefficiency especially with large RTTs and recommended researchers to explore different adaptive streaming mechanisms to optimize both bandwidth usage and the user’s viewing experience. Jiang et al. (2012) tested various commercial HTTP-based adaptive video streaming players. They observed that players were unstable and unfair in bandwidth allocation. They mentioned that problems arose as a result of overlaying the adaptation logic on top of several layers that may hide the true network state. Feedback signals from the network are not a true reflection of the network state. Adaptation logics can only access coarse information from the network state since they run in an application-level sandbox. They suggested that one solution to the problems would be to re-architect the transport layer for video players.

Esteban et al. (2012) investigated the interplay between DASH and TCP CUBIC. They discovered that DASH streams cannot fully utilize the available bandwidth due to the on-off traffic pattern of DASH's interaction with TCP. DASH streams generate sequential HTTP requests every few seconds. After receiving an HTTP request, TCP tries to send the data of the next chunk to the receiver as fast as possible. After transmitting all this data, TCP enters a silent period waiting for the next request. The on-off traffic pattern of this behavior is a challenge for TCP, which performs best when there is a steady stream of data packets and returning ACKs between the end-end points. They concluded that traffic shaping might improve throughput in some cases. Traffic shaping is a technique for spreading the transmission of TCP packets across the entire duration of the estimated RTT rather than sending bursts. They also discovered that long RTTs negatively affect the efficiency of DASH.

Hu et al. (2016) mentioned that the strategies of conventional TCPs might cause user perceived quality degradation in DASH streaming. They proposed increasing TCP's initial congestion window size to alleviate the on-off traffic pattern. They suggested the further exploration of potential improvements to TCP strategies. They observed bit-rate fluctuation and unfair bandwidth sharing when a DASH connection competes against other flows. The results of the study also confirm that long RTTs negatively affect the efficiency of DASH video transfer.

Huang et al. (2012) measured popular video streaming services and observed that accurate client side bandwidth estimation on the HTTP layer is difficult. Rate selection based on inaccurate information can lead to variable and low video quality because of the underestimated fair share of the bandwidth. Instead of improving information flow from TCP to the HTTP layer, they suggest that the DASH client does not attempt to estimate bandwidth at all. The rate adaptation decisions of the DASH client should only maintain the correct receiving buffer level. This behavior yields the correct separation of the protocol layers. Only the TCP protocol is responsible for the objective of obtaining a fair share of bandwidth. Other video streaming studies also describe similar problems as those mentioned above (Akhshabi et al. 2012; De Cicco et al. 2013; Mansy et al. 2013).

The above-mentioned studies also present solutions to these problems. In some cases, these solutions alleviate the problem but raise other issues. For example, the on-off traffic pattern can be moderated by using server-side traffic shaping. However, this creates a complex rate adaptation logic where rate adaption is based on collaboration between the DASH client, TCP, and the traffic shaping module of the server-side application layer.

The problems mentioned indicate that TCP congestion control mechanisms are not optimal for video streaming. The problems are essentially the same in DASH-based streaming as in the traditional one-rate TCP-based streaming: an on-off traffic pattern, sending rate fluctuation, unfairness, bias against long round-trip times, and ineffective use of bandwidth. In DASH-based implementation, another challenge is good co-operation between the TCP protocol and the DASH rate adaption. The main objective of the present study is to develop the dual-mode congestion control mechanism for video services. However, another aim of the study is for the new mechanism to cope better with these problems than TCP's congestion control.

4 Dual-Mode Congestion Control Mechanism for Video Services

In this chapter, the basic algorithm for the dual-mode congestion control mechanism is presented along with the test results of the algorithm. Simulations play a big role in this chapter because the algorithm was developed and first tested using simulations. The real network tests were executed subsequently.

To recap, the goals of the new congestion control mechanism are as follows:

- The mechanism includes two modes. There is a backward-loading mode for low-priority services in which bandwidth is given to other connections after a certain load level. There is also a real-time mode that demands its fair share of the network bandwidth.
- The algorithm should be receiver-based so that there are not complicated conclusion procedures on the heavy-loaded server side.
- The implementations of the modes should be as convergent as possible. In particular, the sender side should be as similar as possible for both modes, enabling seamless switching between modes.
- The real-time mode should be TCP-friendly but the bandwidth should be used much more smoothly than TCP does.
- The mechanism should be less biased against long round-trip times than TCP.
- The mechanism facilitates DASH-based rate adaptation.
- The mechanism should be stable and scalable.

As an additional target, the avoidance of unnecessary complexity can be mentioned. The developed algorithm is called Congestion control for VIdeo to Home Internet Service (CVIHIS). If we wish to emphasize that a specific issue is related to the backward-loading mode version, we use the abbreviation CVIHIS-BLM. Similarly, for the real-time version, we use the abbreviation CVIHIS-RTM.

4.1. CVIHIS algorithm

Based on the findings of the study by Vihervaara and Loula (2014), this thesis presents an improved version of the algorithm. The paper by Vihervaara and Loula (2015) is related to this improved version. This section presents only the basic algorithm, while all kinds of special cases are bypassed. These special cases include, for example, a special start phase algorithm and an extension dealing with dropped packets due to non-congestion situations. Some elaboration work and extra discussion are presented in Chapter 5.

4.1.1. Basic properties of the CVIHIS algorithm

Sending rates can be controlled in either a rate-based or a window-based manner. It is also possible to use a combined rate- and window-based approach. Cen et al. (1998) introduced this kind of combined solution. One of the targets of the congestion control mechanism presented in this thesis is relative simplicity. Therefore, the combined solution can be excluded and either rate-based control or window-based control must be chosen. The natural choice for CVIHIS, and traditionally the more commonly used approach with multimedia streaming, is the rate-based approach. Akan (2004) notes that the window-based approach is not suitable for continuous multimedia streaming since its ACK-controlled packet injection method does not maintain smooth rate variation. On the contrary, window-based control tends to generate a bursty-like traffic output. Window-based control is especially unsuitable for long round-trip time connections (Wang et al. 2008b).

However, rate-based control also has its drawbacks. Although rate-based control works well most of the time, it has the inherent danger of overflowing network buffers, since the metric being directly controlled is the data rate instead of the amount of data (Cen et al. 1998). This means that the sender reduces its data rate only at the request of the receiver. If the ACK messages sent by the receiver are lost inside the network due to severe congestion, the sender will continue sending at the old rate. In rate-based control, this problem can be solved by using a no-feedback timer. In the window-based approach, this problem does not exist because the sender will automatically stop after the granted window closes.

If the network does not support explicit congestion feedback, the sending rate can be adjusted based on packet losses or delays alone. Both indicators are utilized in the mechanism of this thesis, but the algorithm can be said to be somewhat more delay-based than loss-based. The reason for emphasizing the delay-based approach is that it generates suitable conditions for implementing the background-loading mode (Ros and Welzl 2013). For example, LEBDAT and TCP-LP, introduced in Chapter 3, offer mechanisms such that low-priority applications can withdraw from using the bandwidth after the queuing delay exceeds the predefined target. The basic idea behind the delay-based approach is that CVIHIS, and especially its backward-loading mode, tries to keep the queue level of the router at the level of the target delay. It is worth noting that this target delay may exceed the delay demands of applications. If applications have delay and jitter demands, these have to be satisfied by quality of service mechanisms. CVIHIS is a pure congestion control mechanism, not a quality of service mechanism.

The principles of CVIHIS low-priority behavior can be explained with the help of Figure 4.1. In this figure, the delay areas used in CVIHIS rate control are presented. These delay areas correspond to the queue level of the bottleneck router. The minDelay value corresponds to the situation when the queue is empty. The minDelay value includes only propagation delay components, not queuing delays. The queue is empty as long as the sum of the sending rates is below the capacity of the outgoing link. If the sum exceeds the capacity, the queue starts to fill up. The maxDelay point is detected when a packet drop occurs due to queue overflow. CVIHIS tries to keep the queue at the level of the targetDelay area. When operating in the upper delay areas, CVIHIS decreases its sending rate and correspondingly, when operating in the lower delay areas, CVIHIS increases its sending rate.

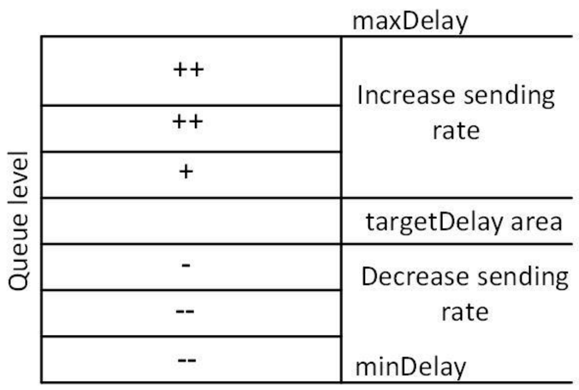


Figure 4.1 Delay areas used in CVIHIS rate adaptation

When sending rates are increasing and the queue level of the router finally reaches the value of the targetDelay area, CVIHIS connections stop increasing their sending rate. However, there may be other kinds of connections that still continue to increase their sending rates. TCP connections react only to packet losses, and these connections will continue to increase their sending rates until the maxDelay point is reached. The UDP protocol can continue to increase its sending rate even after packet drops. These other connections can push the queue level to the upper areas where CVIHIS connections start to decrease their sending rates. With this decreasing behavior, CVIHIS connections using the backward-loading mode make more room for other connections, which can therefore continue to increase their sending rates.

CVIHIS uses one-way delays for delay measurements rather than round-trip times. A one-way delay value is calculated between two network points, and it is the time in seconds that a packet spends traveling across the IP network between these points. The main motivation behind choosing one-way delay is that, in this way, the necessary conclusion procedures can be made on the receiver side. It also has other benefits that are explained by Almes et al. (1999):

- Round-trip measurements actually measure the join performance of two distinct paths. The path from the source to the destination may be different than the path from the destination back to the source (asymmetric paths).
- Even when the paths are symmetric, the paths may have radically different performance characteristics due to asymmetric queuing.

- The performance of an application may depend mostly on the performance of one direction. The applications CVIHIS is designed for are typically this kind. The performance of the data path is more important than the direction in which acknowledgements travel.

There are also drawbacks related to the one-way delay-based mode. The high-resolution clock functions of the operating system must be used. In addition, the clocks of the endpoints have to be synchronized with a considerable degree of accuracy.

In essence, it seems that one-way delay measurements are quite simple. The sender can send a probe packet with a timestamp field. When the receiver receives the packet, it gets another timestamp which corresponds to the receive time of the packet. The difference between these two timestamps is the one-way delay. The main problem of measuring one-way delays is that the clocks of the devices are typically non-synchronized on the Internet. If the clocks of the two nodes are not synchronized accurately enough, one-way delay measurement includes the corresponding one-way delay and the clock offset between the nodes. Even if initially accurately synchronized, two clocks will differ after some time due to clock drift. This clock drift occurs because any two clocks count time at slightly different rates. Due to clock offset and clock drift, one-way delay measurements are challenging.

There are several techniques that can be used to synchronize the clocks of the endpoints (Shin et al. 2011). However, these techniques are not utilized by CVIHIS in order to keep the implementation simple. These synchronization techniques are often either too complicated or too inaccurate for CVIHIS.

In fact, clock offset is not a problem for CVIHIS. CVIHIS probes two delay values, which are presented in Figure 4.1. These delay values are minDelay and maxDelay. It can be said that the minDelay value is the shortest one-way delay value experienced during the lifetime of the connection. The maxDelay value corresponds to the situation in which the queuing delay reaches its maximum value. This happens when a packet is dropped due to the buffer overflow of the router. CVIHIS divides the area between the values of minDelay and maxDelay into several delay areas. CVIHIS can do this correctly if the maxDelay is greater than minDelay even if these delay values are negative due to clock offset. The actual one-way delay measurement related to a certain packet includes this same clock offset and, therefore, the calculated delay is within the minDelay-maxDelay area.

Clock drift can cause problems for CVIHIS. If the measured delay value is increasing due to clock drift, the minDelay value will become outdated. After a certain period of time, the minDelay value no longer corresponds to the actual propagation delay of the connection path. In an extreme situation, the measured delay value including only the propagation delay component may reside closer to maxDelay than minDelay. This means that the connection draws the conclusion that there is an incipient congestion in the network, although the queues of the routers are completely empty. This problem can be solved by updating the minDelay value from time to time. This kind of update operation is presented later in this thesis. Clock drift is not so critical for the maxDelay value because the maxDelay value is updated after every packet drop.

There are different ways to adjust sending rates around the targetDelay area. The rate can be adjusted in a very gentle way by shortening the adjustment steps when getting closer to the target delay level. The LEDBAT algorithm uses this kind of approach. However, Carofiglio et al. (2013) suggest a modification to the LEDBAT method. With a very soft approach, it takes a relatively long time to reach the target delay level and, therefore, reduced efficiency is experienced compared with applications using a constant additive rate increase factor. Based on their observations, they proposed using a more aggressive rate adaptation algorithm near the target delay level. According to these findings, it is preferable to use a slightly different approach for CVIHIS.

The rate adaptation approach used in CVIHIS is presented in Figure 4.1. In this method, the rate adjustment is based on thresholds and rate adjustment areas. The areas just beside the targetDelay area have shorter adjustment steps than other areas. There are two motivations behind this approach. Firstly, CVIHIS wishes to approach the targetDelay area smoothly, but at the same time, it wants to be aggressive enough to compete fairly with other connections. We hope that with this method we can achieve a good compromise between softness and aggressiveness. The second motivation is that the TCP friendliness objective of the real-time mode is pursued in this way. A tailored adaptation factor can be used for each area.

Packet drops are also used to adjust the sending rate of CVIHIS. In Chapter 2, the advantages of AIMD control were discussed. It is worth mentioning that this AIMD behavior is interpreted in a relaxed way in CVIHIS. CVIHIS is not a pure AIMD mechanism because CVIHIS also uses additive decrease components. It can be said that CVIHIS is an AIMD-like mechanism. This kind of AIMD-like mechanism can be described by the following rules:

- In a normal situation, the sending rate is adjusted by small changes.
- In a congestion situation when packets are dropped, the sending rate is adjusted downwards with a large multiplicative step. This behavior ensures that the congestion situation is alleviated quickly enough. This multiplicative step is also used to enhance fairness.
- In a congestion situation, downward steps are more likely for greedy connections than well-behaving connections. This approach also improves fairness between connections.

A multiplicative decrease component has to be added to CVIHIS to implement AIMD control. The natural choice is that the packet drop works as the trigger for the multiplicative decrease step. After the packet drop, CVIHIS waits for 1.5 times as long as the round-trip time before it makes new rate adaptation decisions. This enables CVIHIS to wait for the multiplicative decrease to affect the queue level. CVIHIS behaves with respect to the multiplicative decrease step like TCP NewReno, which decreases its sending rate only once per round-trip time.

4.1.2. Backward-loading mode and improving the algorithm by using delay gradients

The algorithm measures the one-way delays of the connection path. With the help of the minimum and maximum delay values, the delay space is divided into rate adaptation areas, as presented in Figure 4.1. By means of these rate adaptation areas, CVIHIS-BLM strives to enter the targetDelay area and stay there. If it can remain there for a long time, it also means that the sending rate has been stabilized to the capacity of the outgoing link because the queue neither grows nor shrinks. For each received data packet, an acknowledgement packet with the rate adaptation command is sent back to the sender.

Unfortunately, the algorithm presented in Figure 4.1 does not stabilize in the targetDelay area and at the target rate. Instead, the sending rate oscillates, as shown in the simulation graph of Figure 4.2. In this simulation, the capacity of the bottleneck link is 600 kbps and the sending rate oscillates around this capacity rate.

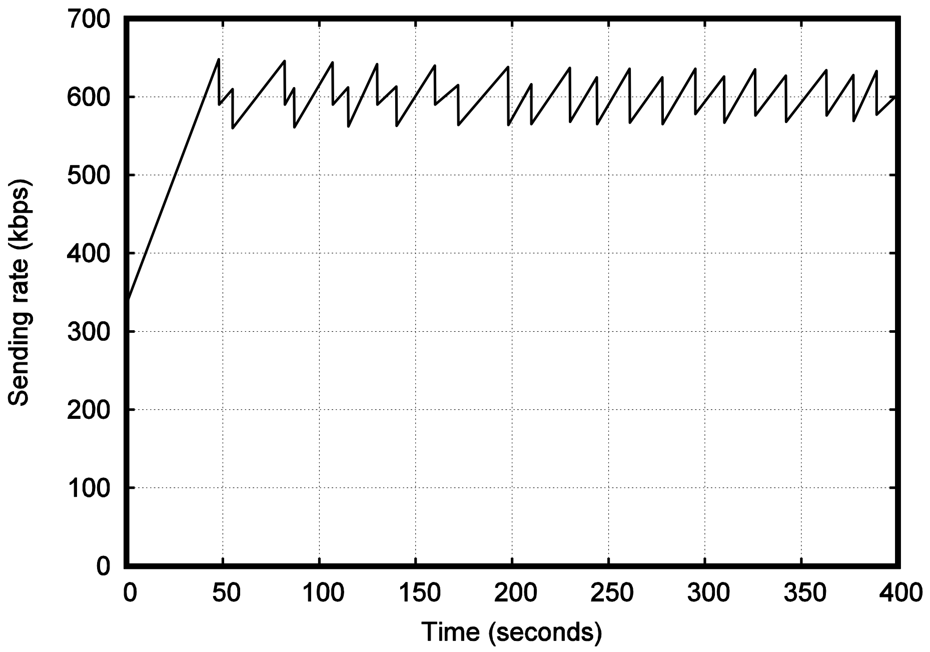


Figure 4.2 Oscillating sending rate of the pre-version

The oscillating behavior can be explained as follows. The sending is started using a moderate rate and the queue does not fill up at the beginning. Queuing is not necessary because the sending rate is below the capacity of the outgoing link. The queue starts to fill up after the capacity of the bottleneck is reached. At the same time, the measured delay values start to exceed the minimum delay. Since the delay values are still below the targetDelay area, the sending rate continues to grow. Even after the target area is reached, the sending rate still continues to increase for a while because it will take some time before the acknowledgement for the data packet that first reached the targetDelay area reaches the sender. All data packets ahead of that particular packet, either in the connection path or in the queue of the router, continue to increase the sending rate. For these reasons, the capacity of the bottleneck link is exceeded, as shown in Figure 4.2.

The reverse phenomenon occurs when the sending rate decreases. The queue does not begin to shorten before the sending rate falls below the capacity of the bottleneck. This is also the moment when the length of the queue reaches its highest value. After this moment, the queue level still exceeds the targetDelay area. Therefore, the sending rate continues to decrease. A similar phenomenon happens again as in the increasing phase, and the capacity is now underestimated. This phenomenon continues, leading to oscillating behavior. The oscillating behavior can be seen more clearly in Figure 4.3. This figure presents the queue size of the bottleneck link. Similar oscillating behavior was mentioned by Wang et al. (1992).

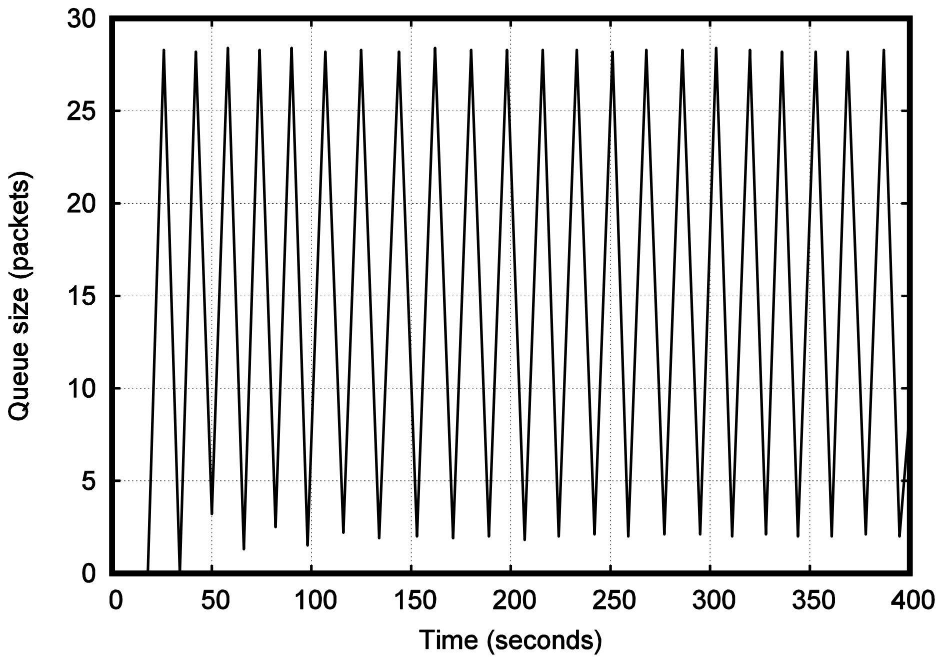


Figure 4.3 Oscillating queue size of the pre-version

The oscillating behavior can be softened significantly by using delay gradients for rate adaptation decisions too. The enhanced and final rate adaptation schema is presented in Figure 4.4. The minimum and maximum delays are determined on the basis of delay measurements. As described above, the minDelay value corresponds to a situation in which the queues of the connection path are empty. The minDelay value includes only propagation delay components. The maxDelay point is found when a packet drop occurs due to queue overflow. The delay value of the last received packet before the dropped packet is used for the maxDelay value because the delay value of a dropped packet cannot be measured.

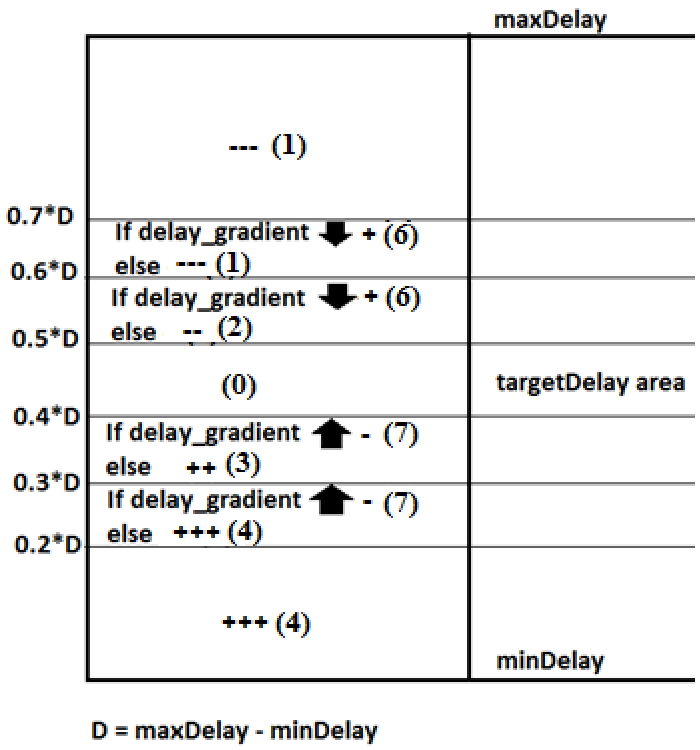


Figure 4.4 Delay areas and rate adaptation steps of CVIHIS

With the help of the minimum and maximum delay values, the delay space is divided into seven rate adaptation areas. It could also be stated that the queue of the router is divided into corresponding parts. The seven rate adaptation areas are used so that sufficiently accurate information about the state of the network can be given to end hosts. Based on theoretical analysis and simulations, Qazi and Znati (2011) note that 3-bit congestion control feedback is sufficient for achieving near-optimal rate convergence to an efficient bandwidth allocation. While the performance gap between 2-bit and 3-bit schemes is large, gains follow the law of diminishing returns when more than 3 bits are used. The integer values in round brackets refer to the rate adjustment commands of CVIHIS that are sent from the receiver to the sender.

The central delay area, called the targetDelay area, is not located in the middle of the delay space but is shifted somewhat downwards so that queues can be kept short. The displacement used in this case is perhaps too modest for normal operating conditions in practice. However, if routers are equipped with active queue management mechanisms, this displacement is perhaps appropriate. The locating factors for each delay area are presented on the left-hand side of Figure 4.4.

There is a black arrow inside some delay areas. This black arrow represents the direction of the delay obtained by comparing the delay values of two consecutive packets. If the arrow points upwards, delays are increasing and the delay gradient is positive. This also means that the queue is filling up. If the arrow points downwards, delays are decreasing and the delay gradient is negative, indicating that the queue is emptying. There are four delay areas with black delay gradient arrows. Inside these four areas, the rate adaptation command is based on the actual delay value and the value of the delay gradient. The delay value assigns a certain delay area. Inside this delay area, the delay gradient can affect the actual rate adaptation feedback. Two extreme delay areas do not use delay gradients for rate adaptation decisions because these areas are far away from the targetDelay area.

By using this kind of rate adaptation scheme, it is hoped that the targetDelay area can be reached without significant oscillation. It can be said that this rate adaptation scheme tries to achieve two target values. It tries to drive the queue level to the targetDelay area by means of the actual delay value. On the other hand, the adaptation scheme also tries to adapt the sending rate to the level of the bottleneck capacity. This is done by means of the delay gradient. If there is a conflict of interests between these two targets, the sending rate target is favored. For example, let us take a case where the measured delay value is inside the fifth delay area ($0.5D$ – $0.6D$). If, at the same time, the delay gradient is negative (i.e., black arrow points downwards), it means that the queue is emptying and the sending rate is below the capacity of the bottleneck link. In such a situation, CVIHIS-BLM has to increase its sending rate because the rate target is not being met. This rate increase action is taken despite the fact that the queue level exceeds the targetDelay area. On the other hand, if the delay gradient is positive, there is no conflict. Both targets dictate that the sending rate has to be decreased.

In its additive increase phase, the TCP protocol increases its transmission rate by one segment for each round-trip time interval. In its basic form, CVIHIS increases or decreases its transmission rate by one packet for each square root of the round-trip time interval. This selection is based on two arguments. Firstly, in this way CVIHIS alleviates the favoring behavior of short connections. Secondly, this allows CVIHIS to react better to long round-trip times. With the help of simulations, it was found that if the transmission rate is adjusted by too small steps then, in the cases of long RTT connections, CVIHIS-BLM could not always enter the targetDelay area. CVIHIS-BLM was informed about the state of the network with long delay and too small steps were taken to react properly. By adapting the transmission rate proportionally to the square root of the round-trip time, a solution can be reached in which the backward-loading mode version of CVIHIS can stabilize in the targetDelay area. In addition to measuring the one-way delay of data packets, CVIHIS also has to estimate the round-trip time of the connection.

CVIHIS does not use equal increase and decrease steps in all cases. In Figure 4.4, these different size steps are shown by a different number of + or - marks. If there are three marks, CVIHIS increases or decreases its transmission rate by one packet for each square root of round-trip time. If there are two marks, the adjustment steps are shorter. The shortest steps are indicated by using only one + or - mark. To enter the targetDelay area smoothly, CVIHIS uses short steps in the delay areas just beside the targetDelay area (rate adaptation feedbacks 2 and 3). The adjustment steps related to the delay gradients (rate adaptation feedbacks 6 and 7) are the shortest ones. By using the shortest steps for delay gradients, we can prevent CVIHIS-BLM from oscillating over the delay areas just beside the targetDelay area. The longer steps related to rate adaptation feedbacks 2 and 3 can push CVIHIS-BLM away from these delay areas.

According to the design criteria, CVIHIS should be a receiver-based congestion control mechanism. This means that all conclusion procedures presented in this section are implemented on the receiver side. In addition, the sender side of CVIHIS implements the EWMA mechanism for smoothing the sending rate. EWMA also mitigates the effects of atypical situations on the operation of the algorithm. For example, momentary excess delaying in a wireless environment is not critical to the proper operation of the algorithm.

4.1.3. Real-time mode

The algorithm presented so far is suitable for the backward-loading mode because the connections using this algorithm withdraw from using the bandwidth after a certain load level. In contrast, the real-time mode connections of CVIHIS should compete with TCP-based connections for their fair share of the bandwidth. Therefore, the algorithm has to be modified so that it will behave more aggressively in the cases of real-time mode connections. One possibility for achieving TCP friendliness would be to imitate TCP behavior as exactly as possible. This would mean the use of a purely loss-based congestion control approach. CVIHIS takes a different kind of approach, however, so that the implementations of its two modes are as similar as possible.

The real-time mode uses a simple approach in which the minimum delay value is pushed upwards in a continuous manner. The delay areas presented in Figure 4.4 are also pushed upwards and, therefore, CVIHIS behaves more aggressively. This upwards pushing is done only when competing behavior is really necessary. If the last measured delay value is smaller than the pushed minimum delay value, the minimum delay value is set to the value of the last measured delay. The pushing function works in a flexible way.

This type of minimum delay pushing means that the real-time mode version is no longer a pure delay-based congestion control solution. The pushing operation shifts this version somewhat towards loss-based congestion control. CVIHIS-RTM is a hybrid solution, a delay-loss-based solution. If the load of the router increases slowly, the real-time mode works like a loss-based approach. Due to the pushing of the minimum delay value, the queue level corresponds to the lowest delay area of Figure 4.4 regardless of the increase in queue level and CVIHIS-RTM increases its sending rate until a packet drop occurs. After that, CVIHIS-RTM takes its multiplicative decrease step. On the other hand, if the load level of the router increases abruptly in the case when there are already packets in the queue, the queue level can be shifted to the level of the upper delay areas. In this case, CVIHIS-RTM decreases its sending rate according to the delay-based approach. It can be said that CVIHIS-RTM has an ability to alleviate burstiness. It is also advantageous to shift the real-time mode of CVIHIS towards the loss-based approach because TCP's congestion control is loss-based. It is difficult to get different kinds of congestion control approaches to operate smoothly together. CVIHIS-RTM already differs from TCP because CVIHIS uses a rate-based approach whereas TCP uses a window-based approach.

This pushing function makes it possible for CVIHIS-RTM to compete with the TCP protocol. In addition, it alleviates the rerouting problem typical of delay-based congestion control mechanisms. The rerouting problem will be discussed in more detail in the next chapter. Pushing the minimum delay value upwards also helps to cope with the clock drift problem. As mentioned in section 4.1, it is desirable that the minDelay value is updated in some manner so that the clock drift accumulation can be reset from time to time. Due to the clock drift problem, it might also make sense to equip the backward-loading mode with minimum delay level pushing. A small pushing factor could be used with the backward-loading mode.

4.1.4. Finding the right rate adjustment parameters

There are six parameters to tune CVIHIS to operate properly. Proper operation means that the backward-loading mode can enter the targetDelay area and stay there if the back-off operation is not required. It also means that the real-time mode can work in a TCP-friendly way. Suitable parameter groups were searched for and found by means of the same type of test cases as presented at the end of this chapter.

It should be emphasized that these parameters are not user parameters that are configured manually by the user. They are parameters that are determined by the design work of the algorithm. These parameter values can be changed in the future, so that the algorithm can be upgraded to the current network environment. However, the point is that the network topologies and network techniques used have no effect on these parameter values. The values are affected by the prevalent congestion control techniques that compete with CVIHIS for the use of network resources. For example, this thesis uses different parameter groups in simulations and real network tests because they use different versions of TCP. Of course, these tests could use the same parameter group but the TCP friendliness would have suffered somewhat.

Tables 4.1 and 4.2 summarize the values of the suitable parameter groups. Table 4.1 presents the rate adjustment parameters of CVIHIS for simulation tests and Table 4.2 presents the rate adjustment parameters of CVIHIS for real network tests. Some of these parameters differ somewhat from the simulation parameters, because the TCP version used is NewReno. NewReno is slightly more aggressive than Reno, which was used for TCP communication in the simulation tests. The four leftmost parameters have not been changed whereas the three rightmost parameters have been updated. All the three updates make CVIHIS more aggressive.

The integer values in round brackets refer to the rate adjustment commands of CVIHIS presented in Figure 4.4. The four leftmost columns present the cases where the rate adjustment is based on the square root of the round-trip time. The value indicates how many more or fewer packets CVIHIS will send during the next square root of the round-trip time than the one immediately before. These rate adjustment steps can be seen in Figure 4.4. In fact, the first column is not a real *tuning* parameter. This parameter is fixed to 1.0. The delay area just above the targetDelay area uses a bigger value than the area just below the target area. If the queue level exceeds the target level, it is desirable that the queue empties a little faster so that queuing delay can be kept short. MD is a multiplicative decrease factor which is used after packet drops to increase the packet sending gap. SF is a smoothing factor used for EWMA to smooth the sending rate of CVIHIS. PF is a pushing factor used only by the real-time mode. This factor is used to multiply the difference between the current minimum and maximum delay values. The result of this multiplication is added to the current minimum delay value.

Table 4.1 Rate adjustment parameters of CVIHIS for simulations

--- (1) +++ (4)	-- (2)	++ (3)	- (6) + (7)	MD (5)	SF	PF
1.0	0.7	0.5	0.2	1.25	0.4 * last update 0.6 * history	0.02

Table 4.2 Rate adjustment parameters of CVIHIS for real network tests

--- (1) +++ (4)	-- (2)	++ (3)	- (6) + (7)	MD (5)	SF	PF
1.0	0.7	0.5	0.2	1.10	0.5 * last update 0.5 * history	0.05

4.1.5. Pseudocode presentation of the algorithm

In this section, the implementation principles of the CVIHIS algorithm are introduced with the help of a pseudocode presentation. The presentation uses a C programming language-like syntax. This pseudocode presentation uses the values of the rate adjustment parameters presented in Table 4.2 for real network tests.

The sender side pseudocode is:

```
PROGRAM CVIHIS_SENDER

struct CVIHIS_data_packet_t {
                             uint32_t segno;
                             time_t packet_send_time;
                             time_t round_trip_time;
                             char[] data;
                             };

struct CVIHIS_ack_packet_t {
                             uint32_t ack_segno;
                             uint8_t rate_change_command;
                             time_t data_packet_send_time;
                            };

CVIHIS_data_packet_t data_packet;
CVIHIS_ack_packet_t ack_packet;
uint32_t_ segnumber = 1;
time_t send_gap = INITIAL_SEND_GAP;
time_t rtt_time = INITIAL_RTT; //250ms in this study

FUNCTION main()
  uint8_t rate_update_command;
  time_t now, previous_ack_received, updated_gap;
  previous_ack_received = now();
  set_signal_handler(signal_handler_send_packet(),
  SIGNAL_SEND_NEXT_PACKET);
  set_timer(send_gap, SIGNAL_SEND_NEXT_PACKET);

  // A while loop for receiving acknowledgement packets and
  // updating the send_gap variable.
  WHILE(1)
    receive_from_client(ack_packet);
    rate_update_command = ack_packet.rate_change_command;
    now = clock();
    rtt_time = now - ack_packet.data_packet_send_time;
    SWITCH rate_update_command
      CASE 1: updated_gap = 1/((1/send_gap)-(1*(now-
              previous_ack_received)/sqrt(rtt_time)));
              BREAK
      CASE 2: updated_gap = 1/((1/send_gap)-(0.7*1*(now-
              previous_ack_received)/sqrt(rtt_time)));
              BREAK
      CASE 3: updated_gap = 1/((1/send_gap)+(0.5*1*(now-
              previous_ack_received)/sqrt(rtt_time)));
              BREAK
      CASE 4: updated_gap = 1/((1/send_gap)+(1*(now-
              previous_ack_received)/sqrt(rtt_time)));
              BREAK
      CASE 5: updated_gap = send_gap*1.10;
              BREAK
      CASE 6: updated_gap = 1/((1/send_gap)+(0.2*1*(now-
              previous_ack_received)/sqrt(rtt_time)));
              BREAK
```

```
55          CASE 7: updated_gap = 1/((1/send_gap)-(0.2*1*(now-
56                  previous_ack_received)/sqrt(rtt_time)));
57                  BREAK
58          DEFAULT: updated_gap = send_gap*1.000;
59                  BREAK
60        END SWITCH
61        send_gap = (0.5*updated_gap) + (0.5*send_gap);
62        previous_ack_received = now;
63        IF (send_gap > MAX_GAP)THEN send_gap = MAX_GAP; END IF
64      END WHILE
65    END FUNCTION
66
67    // Handler for SIGNAL_SEND_NEXT_PACKET to send the next
68    //packet
69    FUNCTION signal_handler_send_packet()
70      data_packet.packet_send_time = clock();
71      data_packet.segno = segnumber++;
72      data_packet.round_trip_time = rtt_time;
73      // set also the data field of the packet
74      send_to_client(data_packet);
75      set_timer(send_gap, SIGNAL_SEND_NEXT_PACKET);
76    END FUNCTION.
```

The packet header structures of CVIHIS are defined between code lines 3 and 14. The type of *time_t* is an integer variable, which is 32 or 64 bits long. The sender side has two main functions. It sends data packets and receives acknowledgement packets for the rate adjustment. The sending event is realized by calling a signal handler *signal_handler_send_packet()*. On code line 26, it is defined that this handler function is invoked when the signal SIGNAL_SEND_NEXT_PACKET appears. This signal is sent when the timer for sending the next packet is triggered. This signal handler function is presented between code lines 69 and 76. On code line 70, the sending time of the packet is obtained by the *clock* function. Every data packet is identified by the sequence number that is set for the header by code line 71. After this operation, the sequence number variable is incremented by one for the next packet. With the help of the sequence number field, the receiver can recognize dropped packets. The packet is sent by code line 74. Line 75 reschedules the packet sending function. The next sending event will be scheduled after a time gap, which is defined by the variable called *send_gap*. The *send_gap* variable is controlled by the *main* function.

The while loop of the *main* function is executed every time a new acknowledgement packet arrives. This loop is presented between code lines 32 and 64. The main task of this loop is to regulate the sending gap of data packets. This sending gap defines the sending rate of CVIHIS. The *send_gap* variable is updated every time an acknowledgement packet is received. Updating is based on the rate adaptation command issued by the receiver and delivered to the sender by the acknowledgement packet. The rate adaptation command is extracted from the ACK packet on code line 34. The rate adjustment step is defined by an integer number between 0 and 7. The integer values are presented in Figure 4.4 (the integer values in round brackets now refer to the cases of the switch statement). Based on this feedback number, the corresponding rate adjustment step is taken by the switch statement. The switch statement is located between code lines 37 and 60. Integer values from 1 to 4 correspond to the feedback values of the delay areas presented in Figure 4.4. Integer values 6 and 7 correspond to the situations in which the rate adjustment steps are based on delay gradients. The default case of the switch statement corresponds to the targetDelay area whose rate adaptation command is 0. A feedback number of 5 indicates a packet drop situation. A multiplicative rate decrease step is taken after a packet drop by code line 50.

The EWMA mechanism for smoothing the sending rate is realized by code line 61. The maximum sending gap is defined by code line 63. If this maximum gap is used, it can define the minimum rate of CVIHIS. The argumentation for using this minimum rate comes from the application types that CVIHIS is designed for. Video applications usually have minimum rate requirements. This is especially true with real-time applications. It is also good to remember that if an application is sending at a moderate rate, it cannot be the source of congestion.

The receiver side pseudocode is:

```
PROGRAM CVIHIS_RECEIVER

struct CVIHIS_data_packet_t {
                              uint32_t segno;
                              time_t packet_send_time;
                              time_t round_trip_time;
                              char[] data;
                              };

struct CVIHIS_ack_packet_t {
                              uint32_t ack_segno;
                              uint8_t rate_change_command;
                              time_t data_packet_send_time;
                             };
FUNCTION main()
  CVIHIS_data_packet_t data_packet;
  CVIHIS_ack_packet_t ack_packet;
  uint8_t rate_change_command = 4;
  time_t now, p_delay;
  time_t min_delay, max_delay, previous_packet_delay;
  time_t wait_time_after_drop;
  uint32_t acknumber = 1;
  uint32_t last_number_received = 0;
  BOOLEAN CVIHIS_mode;
  BOOLEAN start_phase = TRUE;

  // A while loop for receiving data packets and sending
  // acknowledgement packets
  WHILE(1)
    receive_from_server(data_packet);
    now = clock();
    p_delay = now - data_packet.packet_send_time;
    IF p_delay < min_delay THEN min_delay = p_delay;
    END IF

```

```
        IF start_phase == FALSE THEN
           rate_change_command = 0;
           IF CVIHIS_mode == REAL_TIME_MODE THEN
              min_delay = min_delay + ((max_delay-min_delay)*0.05);
              IF p_delay < min_delay THEN min_delay = p_delay;
              END IF
           END IF
           IF min_delay > max_delay THEN min_delay = max_delay;
           END IF
        END IF

        // Check if a packet is dropped
        IF data_packet.segno > last_number_recived+1 THEN
             max_delay = p_delay;
             start_phase = FALSE;
             IF now > wait_time_after_drop THEN
                rate_change_command = 5;
                wait_time_after_drop = now + 1.5*
                data_packet.round_trip_time;
             END IF
         END IF

        // Delay analyze for the rate adjustment feedback
        IF start_phase == FALSE AND now > wait_time_after_drop
        AND rate_change_command != 5 THEN
            IF p_delay < (min_delay+0.2*(max_delay-min_delay)) THEN
            rate_change_command = 4;
            END IF
            IF p_delay >= (min_delay+0.2*(max_delay-min_delay)) AND
            p_delay <= (min_delay + 0.3*(max_delay-min_delay)) THEN
              IF prevous_packet_delay < p_delay THEN
                 rate_change_command  = 7;
                 ELSE rate_change_command = 4;
              END IF
            END IF
            IF p_delay > (min_delay+0.3*(max_delay-min_delay)) AND
            p_delay < (min_delay+0.4*(max_delay-min_delay)) THEN
              IF previous_packet_delay < p_delay THEN
                 rate_change_command = 7;
                 ELSE rate_change_command = 3;
              END IF
            END IF
            IF p_delay > (min_delay+0.5*(max_delay-min_delay)) AND
            p_delay < (min_delay+0.6*(max_delay-min_delay)) THEN
              IF previous_packet_delay > p_delay THEN
                  rate_change_command = 6;
                  ELSE change = 2;
              END IF
            END IF
            IF p_delay >= (min_delay+0.6*(max_delay-min_delay)) AND
            p_delay <= (min_delay+0.7*(max_delay-min_delay)) THEN
              IF previous_packet_delay > p_delay THEN
                 rate_change_command = 6;
                 ELSE  change = 1;
              END IF
            END IF
            IF p_delay > (min_delay+0.7*(max_delay-min_delay)) THEN
              rate_change_command = 1;
            END IF
        END IF

```

```
97          last_number_received = data_packet.segno;
98          previous_packet_delay = p_delay;
99
100         //Send Ack
101         ack_packet.ack_segno = acknumber++;
102         ack_packet.rate_change_command = rate_change_command;
103         ack_packet.data_packet_send_time =
104         data_packet.packet_send_time;
105         send_to_server(ack_packet);
106       END WHILE
107     END FUNCTION.
```

The rate adaptation feedback value is derived from the delay and loss analysis in the while loop of the *main* function. The analysis is performed every time a data packet is received. On code line 31, the receive time of the data packet is obtained. On line 32, the one-way delay of the packet is calculated. If the calculated delay value is less than the current value of the minimum delay variable, this variable is updated on code line 33.

In the start phase, CVIHIS increases its sending rate until the first packet drop is detected. The *rate_change_command* variable is set to its initial value representing the sending rate increase on code line 18. This variable is updated for the first time when the first packet drop is detected because the code lines between 36 and 95 are ignored in the start phase of the connection.

The default value of the rate adaptation feedback is 0 and this value is put for the *rate_change_command* variable (see code line 37) as soon as the start phase of the connection is over (see code line 50). Only a few lines of code need to be added to realize the minimum delay pushing behavior of the real-time mode. There is no need to change the sender side code. This solution makes it possible to switch between the two modes so that the server side does not need to know which mode is currently being used. The added code lines are presented between code lines 38 and 42. On line 39, the current minimum delay value is updated using a constant factor of 0.05 to multiply the difference between the minimum and maximum delay values. This factor belongs to the adjustment parameters used to tune CVIHIS to be TCP-friendly. A factor of 0.05 was found to work well with the other parameters in the suitable parameter group and is therefore used in the real network test cases of this thesis. Because a pushing operation is done after every received acknowledgement, 0.05 is quite a high value. This high value indicates that CVIHIS must behave quite aggressively in order to be TCP-friendly. In fact, Yang et al. (2001) note that TCP is the fastest protocol to utilize extra bandwidth. It was the most aggressive protocol when four congestion control mechanisms were compared (TCP, TFRC, GAIMD, and TEAR).

A possible packet drop event is observed on line 48 by examining the packet sequence numbers. If a packet drop is detected, the code lines between 49 and 55 will be run. First, the maximum delay value is updated. After the packet drop, CVIHIS waits for 1.5 times as long as the round-trip time before it makes new rate adaptation decisions. This enables CVIHIS to wait until the multiplicative decrease step has had an effect on the queue level. The waiting time is defined on line 53 of the code.

If the packet is not dropped and there is enough time since the last packet drop event, the delay analysis is performed by the code between lines 61 and 94. This analysis is based on the delay areas presented in Figure 4.4. If code efficiency is considered, the use of the if-statement structure is not the best choice because every condition of each if-statement has to be checked. On code lines 97 and 98, the sequence number and one-way delay value of the data packet that has just been processed are placed to corresponding variables.

Immediately after the rate adaptation deduction, CVIHIS sends an acknowledgement message to the sender. The number of the ACK-packet and the value of the just deduced *rate_change_command* variable are placed in the header of the ACK packet on code lines 101 and 102. The transmission time of the received data packet is echoed back to the sender for the round-trip time calculation on code lines 103 and 104. After that, the ACK packet is sent to the sender. After sending the acknowledgement, CVIHIS is ready to receive the next data packet.

4.2. Simulation results

The simulation program used in this thesis is version 2 of the Network Simulator (NS-2). This simulator was chosen because it was the most widely used simulator for network simulations at the time the simulations of this thesis began. However, the NS-3 version is likely to replace it gradually.

NS-2 (NS2 2013) is an open source program running in the Linux environment. Thanks to the open source nature, NS-2 can be expanded and modified to fit custom needs. For example, the original protocol objects can be modified and new protocol objects can be implemented. The new congestion control mechanism developed in this thesis has been implemented using a new transport layer protocol object of NS-2. The output of the NS-2 simulation is a text file called the trace file. After the simulation, useful information must be extracted from the trace file with the help of some text-processing program. In this work, Perl scripts are used for information extraction.

Please note that in the simulations of this thesis, the minimum and maximum delay values are free from misinterpretation. There are no packet losses due to bit errors. In addition, there are no clock offsets and clock drifts present in the simulation environment.

4.2.1. Structure of test network for simulations

The typical structure of the test network for the simulation cases presented in this thesis is shown in Figure 4.5. There are four end nodes numbered 0, 1, 4, and 5. These end nodes correspond to user terminal equipment. Nodes 2 and 3 are core nodes. Because we are dealing with the TCP/IP protocol suite, these core nodes represent IP routers. However, NS-2 does not distinguish between end nodes and core nodes. In NS-2, there are only nodes, and links between them.

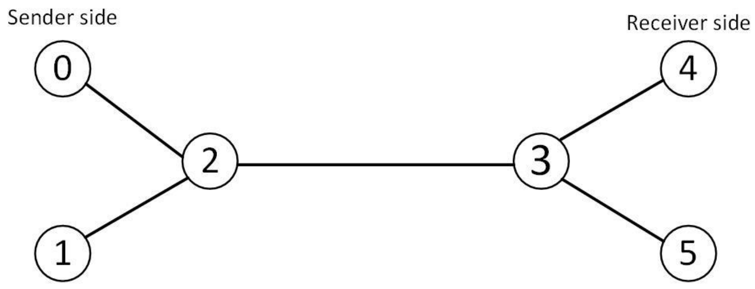


Figure 4.5 Typical structure of the test network

Every end node is connected to a core node by the access link. On the sender side, the capacity of the access link is 1 Mbps. On the receiver side, this capacity is 1.5 Mbps so that there is no bottleneck link at the end part of the connection path. The link between nodes 2 and 3 is a core link. In simulations, this core link can also be called the bottleneck link because its capacity is below the sum of the access links. The capacity of the bottleneck link is a simulation-specific value. At node 2, the queue size for the bottleneck link is also a simulation-specific value ranging from 20 to 50 packets. These values correspond fairly well to the actual values used in real routers with the data rates used. The choice of the queue size is somewhat problematic. Delay-based congestion control is used in this thesis. The queue is divided with the help of thresholds into delay areas and it cannot be too short. There has to be at least a couple of packets inside a certain delay area. If the size of the queue is too large, it takes too much time for the sender to react to the crossing of the threshold. The reason for this is that many other packets have to be de-queued before the threshold packet. This problem is even more serious in relation to the simulations presented in this thesis because only a few connections are present. Many of the queued packets may be the connection’s own packets, which will deliver old state information to the end point. The packet size used is 1000 bytes in all simulation cases. Both CVIHIS and TCP use this same packet size.

4.2.2. Stabilization tests of backward-loading mode

The first objective to be tested is that the backward-loading mode version of CVIHIS can reach a constant sending rate corresponding to the bottleneck capacity if there is no need for the back off operation. This kind of stabilization means that the sending rate does not oscillate. The queue level of the bottleneck should also settle inside the target area. One of the simulation results has been presented in Figure 4.6. As can be seen, the sending rate stabilizes at a constant rate. The constant rate is the same as the capacity of the bottleneck link, i.e., 600 kbps. In this particular case, the average sending rate is 599.9 kbps when the start phase of the connection (up to 80 seconds) is excluded from the calculation. The standard deviation of the rate is 0.5 kbps. The sending rate varies between 598.6 and 600.8 kbps. After the start phase, the rate adaptation commands were mainly zeros indicating no rate changes.

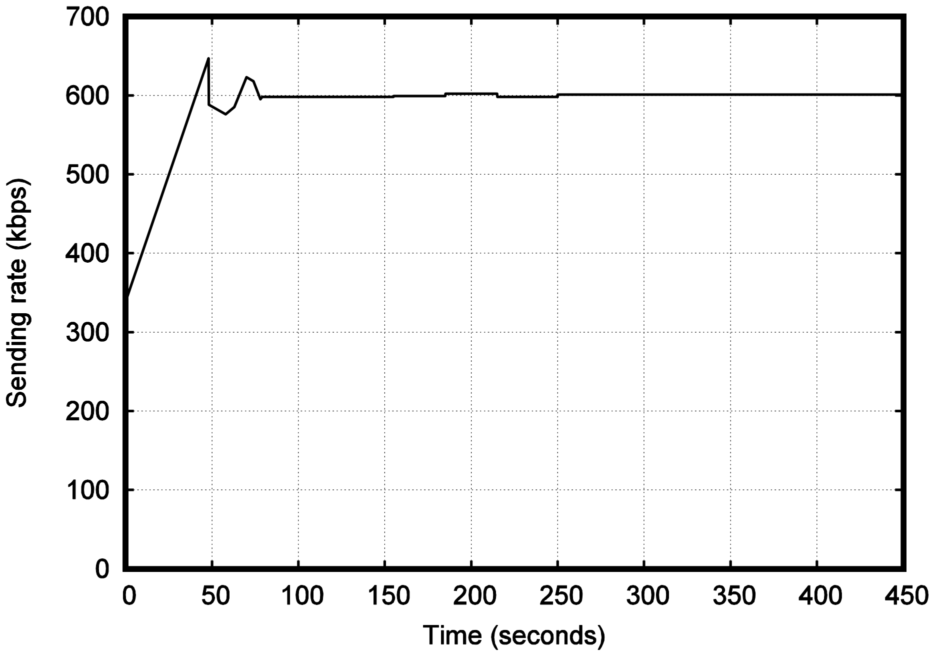


Figure 4.6 Smoothness test of CVIHIS-BLM

Because the basic version of CVIHIS does not have any kind of special start phase algorithm, it takes time to achieve a constant sending rate. There is also some oscillatory behavior at the beginning of the stabilization process. The first rate peak is inevitable because CVIHIS queue level probing does not switch on until a packet drop occurs. The maxDelay value is not defined before the first packet drop. After that, CVIHIS-BLM takes its multiplicative decrease step and the actual stabilization starts. In this simulation case, it takes about 32 seconds before the sending rate stabilizes. 32 seconds is quite a long time in the computer world. On the other hand, the basic algorithm of CVIHIS is optimized for data transmissions lasting at least a few minutes. The specific features of short-term communication can be taken into account by integrating the appropriate start phase algorithm into the basic algorithm.

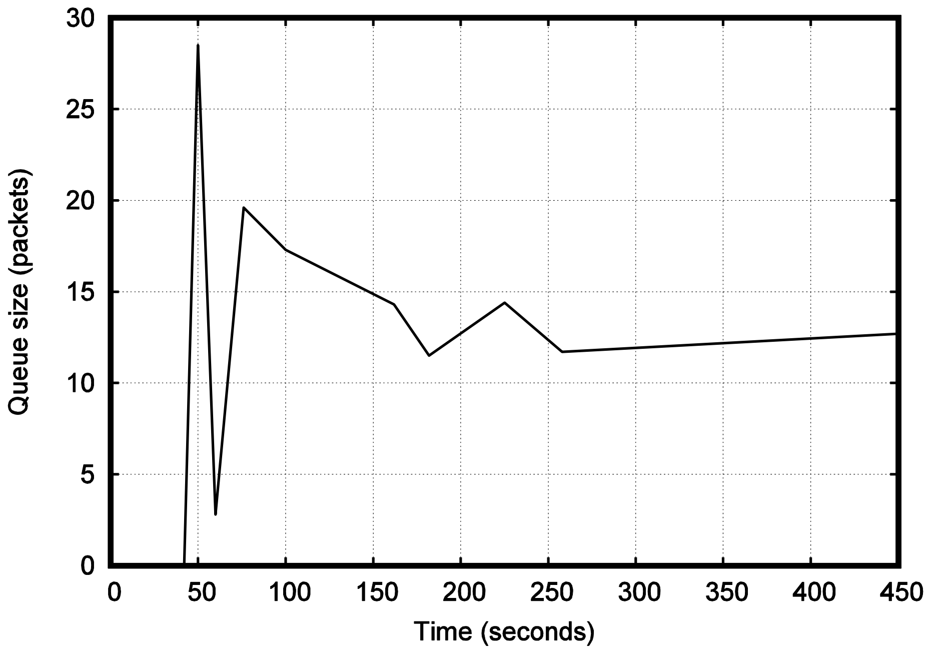


Figure 4.7 Queue behavior of the bottleneck link

Related to this simulation case, the queue behavior of the bottleneck link is presented in Figure 4.7. The queue size of bottleneck node 2 (see Figure 4.5 for the network configuration) is 30 packets in the simulation script. Based on Figure 4.7, it seems that the queue level never reaches its maximum size. In the simulation script, the queue level is saved in 0.2-second steps and, therefore, the script cannot capture the moment of queue overflow. On the other hand, no answer has been found to the question of why the maximum queue length seems to be one package less than the value of the simulation script in all the simulation cases. Taking into account the previous issue related to the maximum queue size, the target queue level of this case is roughly between 11.5 and 14.5 packets and the queue level settles inside this target area.

As can be seen, the queue level fluctuates slowly inside the targetDelay area and its immediate vicinity. This behavior is intentional and was selected from two alternatives. In the first alternative, CVIHIS-BLM could eliminate the fluctuating behavior almost entirely by using the delay gradient investigation inside the targetDelay area. Unfortunately, this investigation would lead to behavior where the sender has to change its sending rate after every rate adjustment feedback. In the real world, and also with the NS-2, the queuing delay cannot be measured as an integer number of packets but is instead a decimal. Therefore, the queuing delay varies all the time as does the delay gradient. By choosing the second alternative, i.e., by letting the queuing delay fluctuate, the sender can send at a constant rate for a long time. Using this alternative, CVIHIS-BLM does not perform any rate adjustment functions inside the targetDelay area. This leads to slightly fluctuating behavior because CVIHIS-BLM cannot enter the target area by using the exact rate of the bottleneck link. If the queue level is supposed to go up or down, the sending rate cannot be the same as the capacity of the link.

Stabilization behavior was ensured by performing twenty simulations. These simulation cases are presented in Table 4.3. The queue size varied between 20 and 50 packets, the capacity of the bottleneck link varied between 500 and 900 kbps, and the one-way propagation delay varied between 30 and 180 milliseconds. The case presented in Figure 4.6 is the seventh case in the table. The results of the twenty simulations were quite consistent. The sending rate stabilized in every case. After the stability point was reached, the standard deviation of the sending rate was below 1 kbps in all the cases. The time needed for stabilization varied between 15 and 61 seconds. This time was defined as the time from the first packet drop until the sending rate stabilized. The stabilization time is presented in the last column of Table 4.3. The longest stabilization times occurred when the one-way propagation delay was long.

Table 4.3 Twenty simulations for testing the stabilization of the backward-loading mode

	Bottleneck capacity [kbps]	Queue size [packets]	One-way delay [ms]	Time for stabilization [s]
1	500	20	30	15
2	500	30	150	15
3	500	40	180	13
4	500	50	60	32
5	600	50	135	30
6	600	40	60	36
7	600	30	90	32
8	600	20	180	40
9	700	30	180	52
10	700	50	30	33
11	700	40	135	49
12	700	20	45	24
13	800	40	75	39
14	800	50	30	30
15	800	50	180	61
16	800	30	60	32
17	900	20	150	31
18	900	40	30	33
19	900	30	90	32
20	900	50	180	59

4.2.3. Back off tests of backward-loading mode

The second objective of CVIHIS-BLM was for it to withdraw from using the bandwidth when the sum of the sending rates of the bottleneck connections exceeds the link capacity for a sufficient amount of time. This means that the queuing delay exceeds the target delay area. It should also take advantage of the free capacity released from other connections. Figure 4.8 presents the result of the simulation in which the back off behavior of CVIHIS-

BLM was tested. CVIHIS-BLM is presented in this figure using red. In this case, the capacity of the bottleneck link is 600 kbps and the one-way propagation delay is 90 milliseconds. The CVIHIS-BLM connection was started at the beginning of the simulation. Then, we waited until the sending rate of CVIHIS-BLM settled at the capacity of the bottleneck link before starting the TCP Reno connection. The TCP connection was activated 120 seconds after the start of the simulation. The TCP connection was stopped after the CVIHIS back off action at 300 seconds. It was checked then that CVIHIS-BLM increased its sending rate again because there was then free capacity in the network.

As far as TCP's sending rate is concerned, there may be some variation generated by the phase effect of the analysis tool. Packets are grouped in order to calculate the sending rate of TCP. This grouping interval is 0.2 seconds and is often not compatible with packet interval times. As a result, there may be differences of one packet between the groups. A packet size of 1000 bytes (8000 bits) is used. If the packet size is divided by the grouping interval of 0.2 seconds, a rate difference of 40 kbps is obtained between the groups. This kind of phase effect was also present in the other simulation cases of this thesis.

This simulation shows that the backward-loading mode works as expected. After the start of the TCP connection, CVIHIS-BLM starts its back off action. This back off action lasts for about 100 seconds. At the end of the back off phase, CVIHIS-BLM reaches its minimum sending rate of 100 kbps. After stopping the TCP connection, CVIHIS-BLM starts to increase its sending rate. It takes as long as 80 seconds to reach the desired rate because EWMA is used to smooth the sending rate.

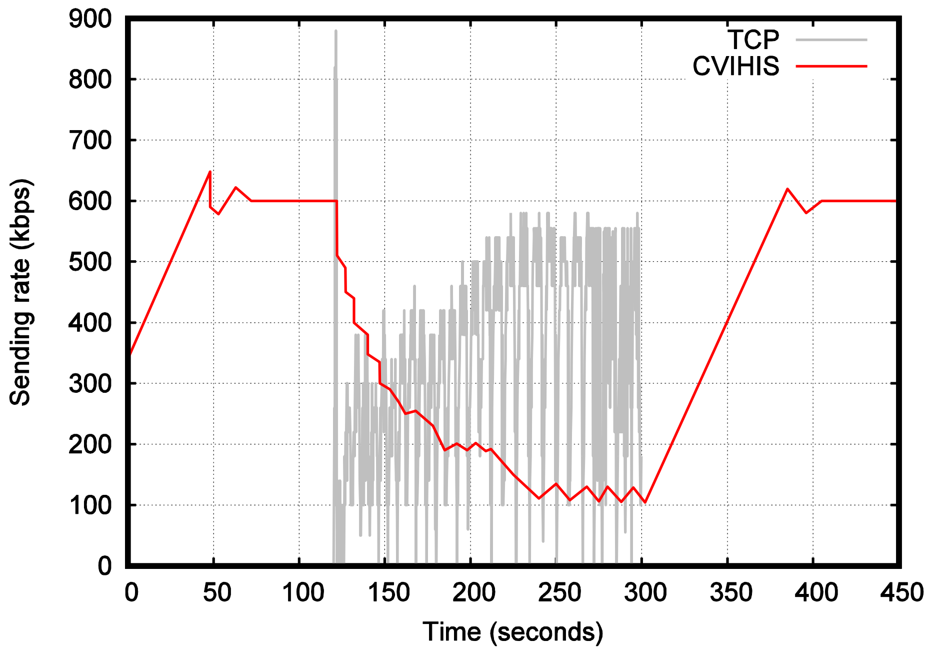


Figure 4.8 Back off test of the backward-loading mode

Figure 4.9 presents the result of a simulation in which the back off behavior of CVIHIS-BLM was tested using another type of experiment. Two TCP connections were present in this case. The network setup is similar to that of the previous case. The aim of this simulation was to check whether CVIHIS-BLM increases and decreases its sending rate as expected. As can be seen, CVIHIS-BLM increases its sending rate when no active TCP connections are present in the network. CVIHIS-BLM also backs off when there is TCP traffic.

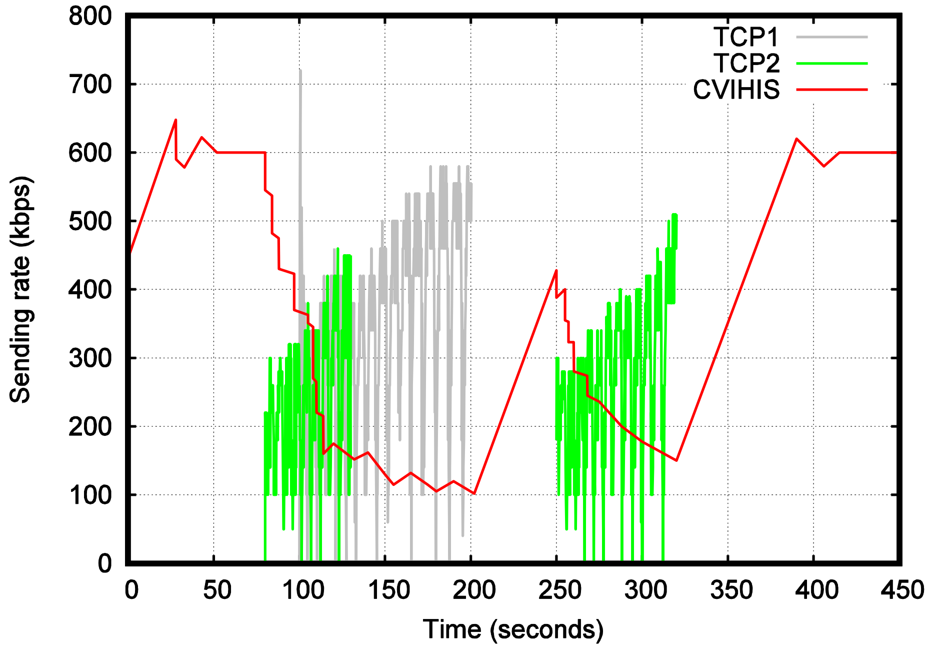


Figure 4.9 Back off test of the backward-loading mode against two TCP connections

The back off behavior was ensured by doing twenty extra simulations using the same network setups as presented in Table 4.3. These simulations were done for one TCP connection as in the case of Figure 4.8. The results of the simulations were consistent with the simulation case presented in Figure 4.8. It was observed that CVIHIS-BLM can adapt its sending rate faster if round-trip times are short. CVIHIS obtains rate feedbacks more frequently when the round-trip time is short. Proper back off behavior was observed in all the cases. Therefore, the conclusion can be drawn that the backward-loading mode behaves as desired.

4.2.4. Basic operation of real-time mode

The aim of CVIHIS-RTM is that the bandwidth is utilized efficiently. In Figure 4.10, the performance of the real-time mode of CVIHIS without other connections in the network is illustrated. The capacity of the bottleneck link is 600 kbps. CVIHIS-RTM is able to take advantage of almost the entire bandwidth. However, the aggressiveness of CVIHIS-RTM does not come without a drawback. As shown in Figure 4.10, the smoothness of the real-time mode suffers because of this aggressiveness. There is a natural explanation for this moderate level of smoothness. Since the pushing function shifts the delay areas upwards, packet drops are generated. As a result of these packet drops, multiplicative decrease steps are activated and thus the smoothness suffers.

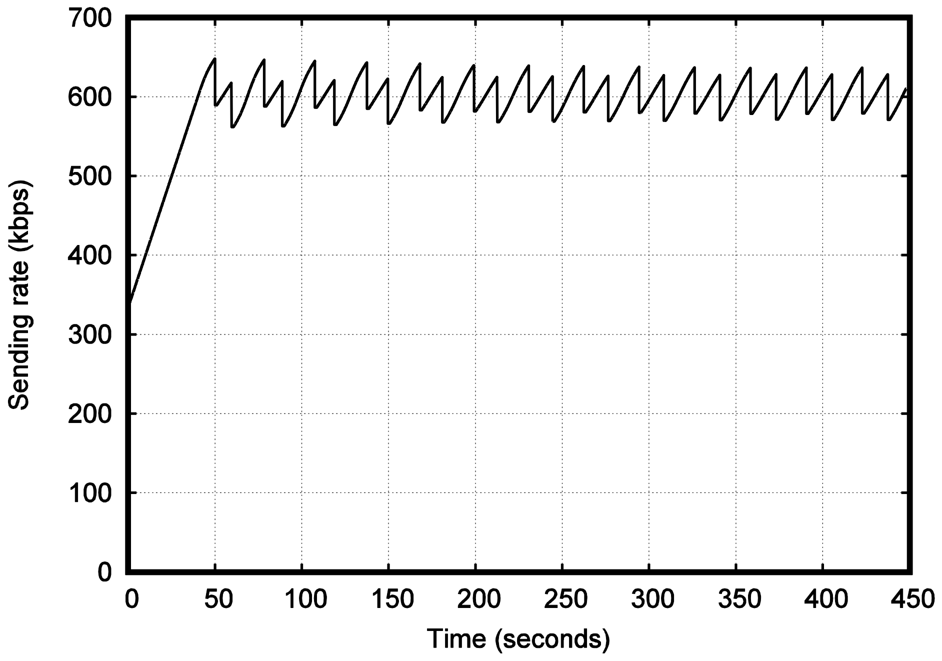


Figure 4.10 Real-time mode with minimum delay value pushing

4.2.5. CVIHIS real-time mode against itself

The CVIHIS-RTM connections should share bandwidth with each other in a fair manner. The sending rates should not differ significantly even though the connections have somewhat different round-trip times because CVIHIS is not strongly biased against long round-trip times. Two simulations were performed to observe the real-time mode behavior against itself. In these simulations, two CVIHIS-RTM connections were concurrently present in the network. Both connections started sending at the beginning of the simulation and the starting rates were equal. However, one of the connections was stopped during the simulation and started again later. The capacity of the bottleneck link was 600 kbps and the queue size 30 packets. The simulation result of the first case is presented in Figure 4.11. The connections have an equal one-way propagation delay of 90 ms. In this simulation, CVIHIS-RTM behaves perfectly. The average rates of both connections are almost equal as soon as the sending rates stabilize.

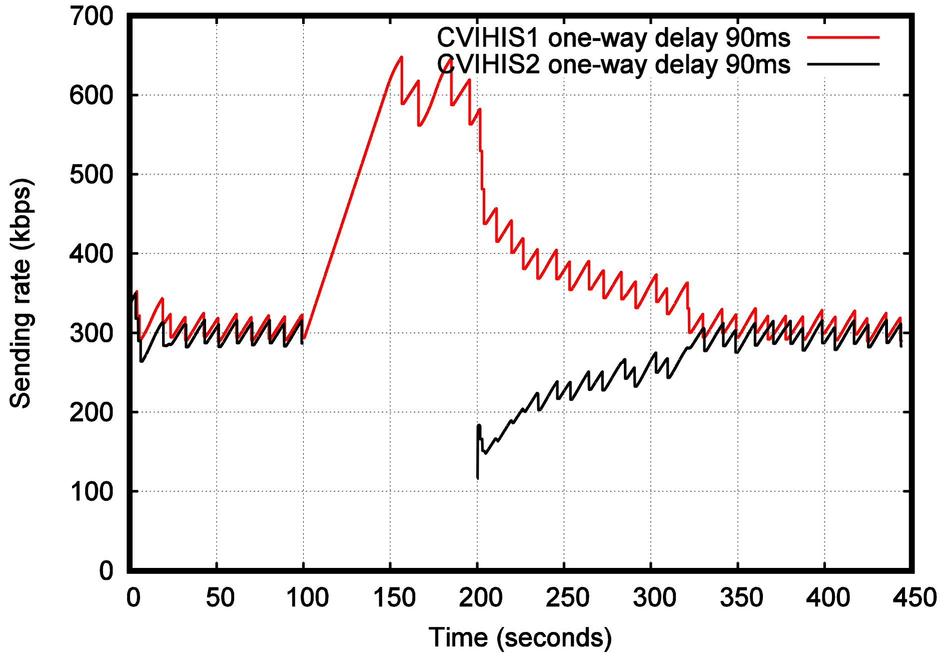


Figure 4.11 CVIHIS-RTM against itself using equal one-way delay values

In the second case, the connections use different one-way delay values. The aim of the simulation was to test whether CVIHIS-RTM favors the connections of shorter round-trip times because CVIHIS increases its sending rate based on the square root of the round-trip time. It is known that TCP owns this kind of favoring property. The simulation results are presented in Figure 4.12. The connection illustrated by the red line has a one-way propagation delay of 50 ms while the other connection has a one-way propagation delay of 150 ms. The simulation shows that these connections stabilize at the same sending rate.

CVIHIS alleviates bias against long round-trip time connections because CVIHIS also takes additive decrease steps when the queue level exceeds the targetDelay area. If the connection can increase its sending rate more during the rate increase phase due to the short round-trip time, it must also take bigger rate decrease steps during the rate decrease phase. The multiplicative decrease step also improves fairness among connections. Based on this simulation, an initial conclusion can be made that CVIHIS does not have significant bias against long round-trip time connections. Even connections with longer round-trip times can compete for the use of bandwidth quite fairly. The results related to the real network tests will present more results related to CVIHIS behavior against itself.

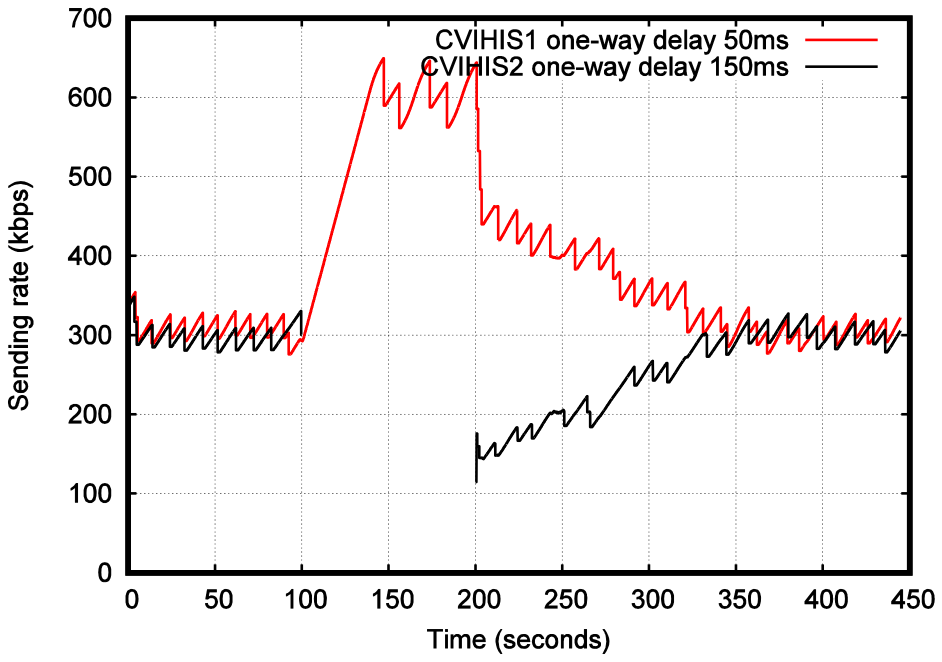


Figure 4.12 CVIHIS-RTM against itself when using different one-way delay values

4.2.6. Testing TCP friendliness of real-time mode

The testing of the TCP friendliness of CVIHIS-RTM is described in this section. The aim of CVIHIS-RTM is good average TCP friendliness. In the worst cases, the objective is that the sending rates of CVIHIS-RTM and TCP are within a factor of two from each other. Compared to the TCP protocol, CVIHIS-RTM should also alleviate unfairness related to heterogeneous round-trip times. At first, CVIHIS-RTM was tested against one TCP connection using three different initial rates. In one of the tests, CVIHIS-RTM started by using an initial rate, which was half of the capacity of the bottleneck link. With the help of this simulation case, it was also tested to see whether the dropping behavior of CVIHIS-RTM was acceptable. In the two other cases, the start rate of CVIHIS-RTM would be below and above a fair share of the capacity. It is worthy of note that TCP does not use any kind of initial rate. TCP starts by using the slow start algorithm by sending only one segment at first. After these three cases, more simulation cases will be run for a more comprehensive evaluation of CVIHIS-RTM's TCP friendliness. There will also be a couple of simulations in which two or three TCP connections are present with CVIHIS-RTM connections.

It should be noted that no official definition for TCP-friendly behavior exists. It is also impossible to get different kinds of mechanisms to work in a friendly way all the time. Floyd and Fall (1999) state that it is to be expected that a TCP-friendly flow would acquire the same bandwidth share as a TCP connection only when averaged over time intervals of several seconds or even over the entire life time of the flow, and not at every time point.

4.2.6.1 Tests against one TCP connection

The testing of the TCP friendliness of CVIHIS-RTM against one TCP connection is described in this subsection. The aim is for CVIHIS-RTM to acquire its fair share of the bandwidth. In addition, its sending rate should not fluctuate strongly. CVIHIS-RTM is presented in the figures in red. In the first simulation case, CVIHIS-RTM started with an initial rate of about 300 kbps. This rate is the fair share rate of CVIHIS-RTM because the capacity of the bottleneck link is 600 kbps. The queue size of the bottleneck node is 30 packets and the one-way propagation delay is 90 ms. The duration of the simulation case was over 400 seconds, similar to most of the simulation cases of this thesis. The simulation result is depicted in Figure 4.13.

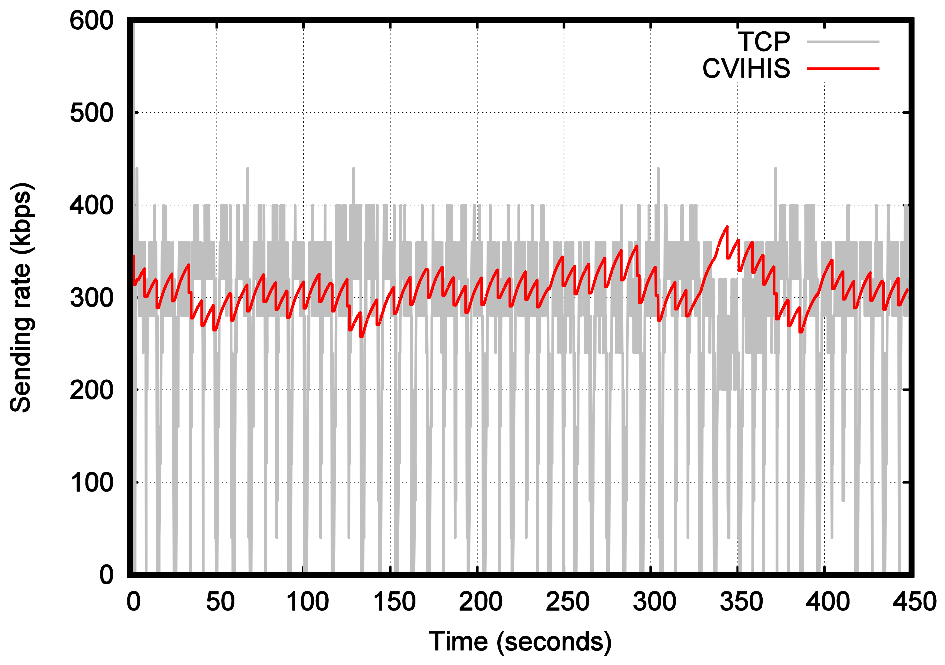


Figure 4.13 CVIHIS-RTM against one TCP connection (CVIHIS initial rate 300 kbps)

As can be seen, the rate of the TCP connection varies much more than the sending rate of CVIHIS-RTM. One reason for this is that TCP uses a much more aggressive multiplicative decrease step than CVIHIS as TCP at least halves its sending rate after a packet drop. Another reason for the high rate fluctuation of TCP is that the TCP version used in this simulation was Reno. The TCP Reno version can take several multiplicative decrease steps during a short period. The NS-2 simulator shows that there is some clustering related to the packet drops. It is to be expected that the TCP sending rate would vary in a somewhat smoother way when testing CVIHIS-RTM against the TCP NewReno version. TCP NewReno behaves regarding clustering dropping like CVIHIS. Neither of these congestion control algorithms take several multiplicative decrease steps during a single round-trip time.

CVIHIS-RTM coped very well in this simulation case. It took almost exactly its fair share of the capacity. The average sending rate of CVIHIS-RTM was 307 kbps while the average sending rate of TCP was 273 kbps. The standard deviation of the CVIHIS-RTM sending rate was 20 kbps. As can be seen, the rate variation of CVIHIS-RTM is much smoother than that of TCP. However, there is still some rate variation related to CVIHIS-RTM in spite of using EWMA to smooth the sending rate. This variation is evident because CVIHIS-RTM takes multiplicative decrease steps and CVIHIS-RTM packet drops were present in this simulation. It is not discussed in this thesis if this level of rate variation is acceptable as it depends on the demands of real applications.

With the help of this simulation case, the packet dropping behavior of CVIHIS-RTM can also be compared with that of a TCP connection when the normal tail-drop scheme is adopted in the bottleneck router. Figure 2.14 presents the cumulative number of dropped packets of both connections. It can be seen that the slow start phase of the TCP protocol can significantly overestimate the capacity of the bottleneck link. At the beginning of this simulation case, CVIHIS and TCP dropped lots of packets.

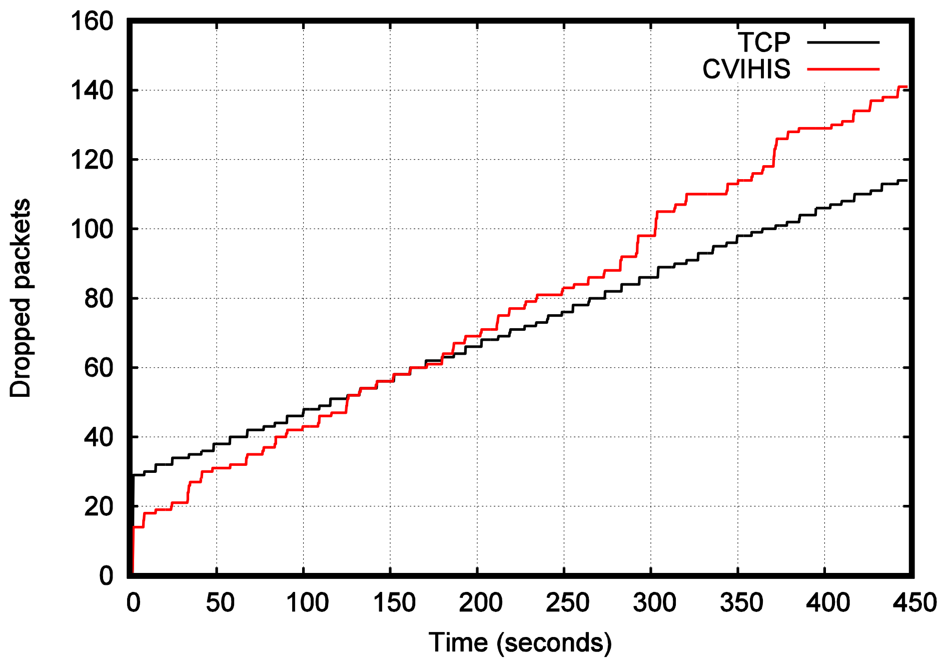


Figure 4.14 Cumulative packet dropping behavior of CVIHIS-RTM and TCP

The dropping rates of the connections differ slightly after the slow start phase. After taking into account the rate differences of the connections, the conclusion can be made that CVIHIS-RTM drops somewhat more packets than TCP. The packet loss rate of CVIHIS-RTM is 0.72%, compared with 0.55% for TCP. This difference in value drops the perceived video quality from 4.2 to 4.0 if the quality is assessed on a scale of 0-5 (Paudyal et al. 2014). The reason for this slightly different dropping behavior is the smoothed sending rate of CVIHIS. TCP leaves the congested situation more aggressively than CVIHIS. Therefore, CVIHIS must sometimes drop more than one packet due to a single congestion situation. These results related to the dropping behavior of CVIHIS-RTM were confirmed using some other simulation cases as well.

In the following simulation case, CVIHIS-RTM started with an initial rate of about 120 kbps. The aim was for CVIHIS-RTM to acquire its fair share of the bandwidth despite the fact that this rate is significantly under the fair share of CVIHIS-RTM. The simulation result of this case is presented in Figure 4.15. The simulation shows that CVIHIS-RTM can reach its fair share of the capacity. The average sending rate of CVIHIS-RTM was 289 kbps while that of TCP was 288 kbps. The standard deviation of the CVIHIS-RTM sending rate was 23 kbps. The calculation of these rates was started at a time unit of 50 seconds. The beginning phase of the simulation was omitted because these connections use different starting rates. Later, other similar cases will be treated in the same way.

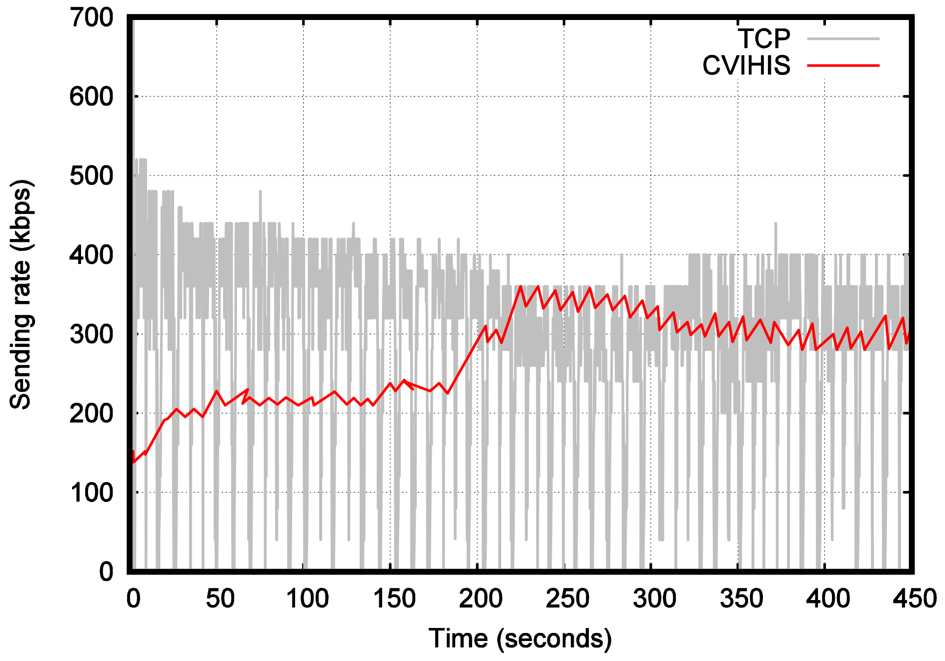


Figure 4.15 CVIHIS-RTM against one TCP connection (CVIHIS initial rate of 150 kbps)

In the next simulation case, CVIHIS-RTM used an initial rate of about 500 kbps. This rate is over the CVIHIS-RTM fair share. The hope was that CVIHIS-RTM would give up some of its sending rate. The result of this simulation case is presented in Figure 4.16. Almost immediately after the beginning of this simulation, CVIHIS-RTM started its back off action. After 50 seconds, the CVIHIS-RTM sending rate settled to a level of about 300 kbps. The average sending rate of CVIHIS was 305 kbps while that of TCP was 276 kbps. The standard deviation of the CVIHIS-RTM sending rate was 23 kbps. In summary, it can be said that CVIHIS-RTM behaves in an acceptable way when competing for bandwidth against a TCP connection.

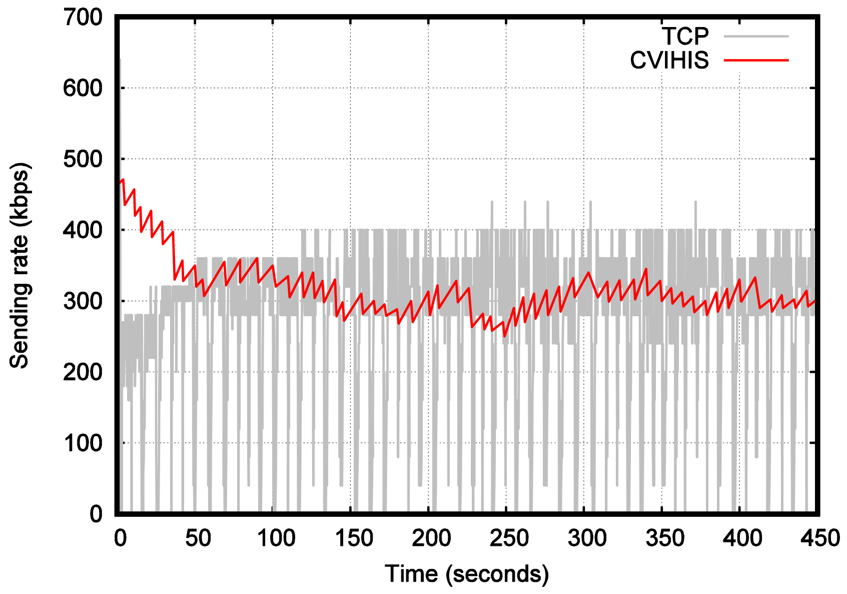


Figure 4.16 CVIHIS-RTM against one TCP connection (CVIHIS initial rate of 500 kbps)

In Figure 4.17, results of the simulations are presented where CVIHIS-RTM competed against one TCP connection in the cases of three different one-way propagation delay values. The delay values were 60, 120, and 180 ms. The queue size of the bottleneck link was 40 packets and the capacity of the bottleneck link varied between 0.5 and 1.4 Mbps. The curves indicate the proportion of the capacity of the bottleneck link acquired by CVIHIS-RTM. The sending rates indicate a good level of averaged fairness. However, there are differences in the sending rates and the average of Jain's fairness indices is 0.986. If the worst cases are considered, a faster connection obtains about 1.5 times as much bandwidth as a slower connection.

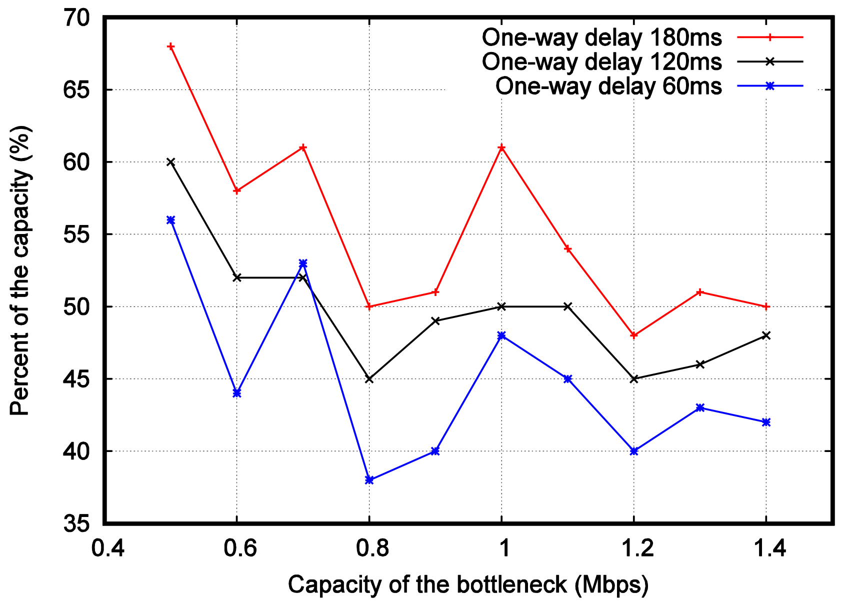


Figure 4.17 CVIHIS-RTM proportion of the capacity against one TCP connection when using three different one-way delay values

The TCP friendliness of CVIHIS-RTM was further studied by performing twenty extra simulations against one TCP connection. In these simulation cases, the queue size varied between 20 and 50 packets and the capacity of the bottleneck link varied between 500 and 900 kbps. It was observed that there might be problems with TCP friendliness if the capacity of the outgoing link was low compared to the sending rate of the CVIHIS-RTM connection. In cases like these, CVIHIS-RTM can dominate the link because CVIHIS-RTM decreases its sending rate in short steps. For this reason, there is too little free capacity for TCP to increase its sending rate. This kind of problem is not serious in real world cases because nowadays one connection cannot saturate the capacity of the router. Due to this finding, the capacity of the bottleneck link was at least 500 kbps in these twenty simulation cases. The one-way propagation delay varied between 30 and 150 milliseconds. The simulation cases also differ from one another in the starting points and starting rates of the connections.

Table 4.4 Twenty simulations for testing the TCP friendliness of CVIHIS-RTM

	Capacity [kbps]	CVIHIS start rate [kbps]	Queue size [packets]	One-way delay [ms]	CVIHIS [%]	CVIHIS deviation [kbps]	TCP [%]
1	500	300	50	75	58	13	42
2	900	300	40	150	45	27	55
3	700	300	50	60	52	24	48
4	600	300	40	120	52	13	48
5	800	300	20	135	48	42	52
6	700	300	20	90	39	26	61
7	500	300	40	60	58	27	42
8	900	300 (TCP -100s)	40	150	44	34	56
9	800	300 (CVIHIS -20s)	35	150	47	32	53
10	600	300 (CVIHIS -15s)	30	30	40	32	60
11	600	300 (CVIHIS -25s)	20	150	54	31	46
12	600	300 (TCP -20s)	20	90	51	26	49
13	800	500	40	75	43	48	57
14	500	500	30	45	46	29	54
15	700	500 (CVIHIS-10s)	25	150	55	33	45
16	900	500 (TCP -20s)	45	60	45	50	55
17	800	150	30	135	43	21	57
18	500	150	45	45	53	19	47
19	700	150 (CVIHIS -10s)	25	90	40	15	60
20	900	150 (TCP -20s)	50	135	46	40	54
Average					48		52

The twenty simulations are presented in Table 4.4. The second column presents the capacity of the bottleneck link. In the third column, the round brackets indicate if and how much the start of one of the connections was delayed with respect to the beginning of the simulation. The actual simulation results are presented in the last three columns. The columns indicate the proportions of CVIHIS-RTM and TCP of the combined sending rate of these two connections as well as the standard deviation of the CVIHIS-RTM sending rate. The results show a good level of fairness on average: CVIHIS-RTM receives 48 percent and TCP 52 percent of the combined rate. The average of Jain's fairness indices is 0.986, indicating that there are differences in the sending rates. The worst cases are perhaps more interesting than the average rates. In the worst case, the faster connection obtains about 1.5 times as much bandwidth as the slower connection. This level of unfairness can be considered acceptable, although there is always some room for personal opinion as TCP fairness is not a well-defined concept.

4.2.6.2 Heterogeneous round-trip times

As equation 2-7 presents, TCP's throughput degrades with higher RTTs. Therefore, a protocol which complies with TCP's throughput has to be biased against high-RTT connections (Widmer et al. 2001). In other words, this TCP-friendly protocol has to favor the connections of short round-trip times. In section 4.2.5, it was tentatively observed that CVIHIS-RTM did not have this notable bias. Some simulations were executed to study this phenomenon more, using different one-way propagation delay values. In each case, there was one CVIHIS-RTM connection against one TCP connection so that the one-way delay values of these connections were equal. The one-way delay values varied from 15 ms to 180 ms in 15 ms steps. The capacity of the bottleneck link was 600 kbps and the size of the bottleneck queue was 30 packets. The simulation results are presented in Figure 4.18.

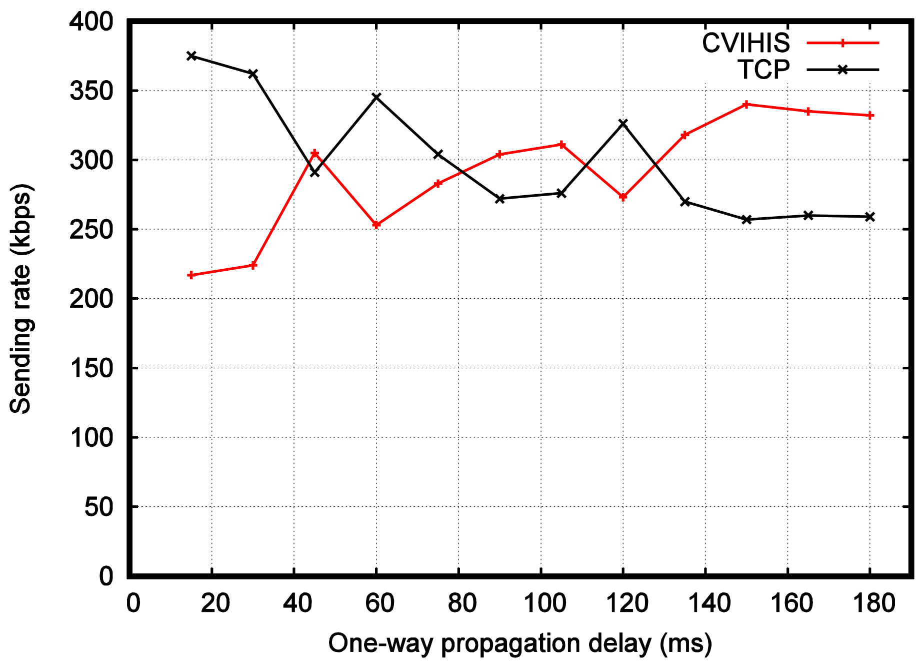


Figure 4.18 CVIHIS-RTM against TCP with different one-way propagation delay values

The simulation results confirm that different round-trip times affect the sending behavior of CVIHIS-RTM much less than they affect that of the TCP protocol. CVIHIS-RTM manages modestly well when round-trip times are very short. In the mid-delay area, the sending rates are quite equal. In fact, the parameters of CVIHIS-RTM are tuned so that it can compete against TCP fairly in this mid-delay area. When round-trip times are long, CVIHIS-RTM manages somewhat better than TCP. These findings are in line with Figure 4.17. Figure 4.18 also shows that the phase effect influences the simulation results. Individual measurements may depart from the trend.

The fact that CVIHIS is not so sensitive to the effects of different round-trip times also has certain advantages. CVIHIS can alleviate unfairness related to heterogeneous round-trip times. This feature is illustrated in Table 4.5, which presents the simulation results of three cases in which the connections have different one-way propagation delay values. The first and second columns present the delay values while the third and fourth columns present the sending rates of these connections. As can be seen, TCP behaves against itself more unfairly than CVIHIS-RTM does against TCP.

Table 4.5 Effect of heterogeneous round-trip times on sending rates

One-way propagation delay		Sending rate	
CVIHIS-RTM 30 ms	TCP 110 ms	CVIHIS-RTM 332 kbps	TCP 271 kbps
CVIHIS-RTM 110 ms	TCP 30 ms	CVIHIS-RTM 249 kbps	TCP 337 kbps
TCP1 30 ms	TCP2 110 ms	TCP1 416 kbps	TCP2 172 kbps

4.2.6.3 Tests against many TCP connections

In the simulation cases described in this section, the TCP friendliness of CVIHIS-RTM was tested using more than two connections in the network. In two cases, one CVIHIS-RTM connection competed against two TCP connections. In these cases, the CVIHIS-RTM connection was sending continuously while the TCP connections were started and stopped during the simulation. The capacity of the bottleneck link was 900 kbps. The queue size of the bottleneck node was 30 packets and the one-way propagation delay 90 ms. As there were two other connections present, 300 kbps is a fair share of the CVIHIS-RTM connection. In these simulations, the capability of CVIHIS-RTM to reduce its sending rate after the start of the TCP connections and to increase its sending rate after the TCP connections have stopped was investigated.

The simulation results of these two cases are presented in Figures 4.19 and 4.20. The simulation cases are otherwise identical except for the lifetimes of the TCP connections. In both cases, CVIHIS-RTM reduced its sending rate after the start point of the TCP connections and increased its sending rate after the end of the TCP connections. In Figure 4.19, CVIHIS-RTM behaves almost perfectly. However, in the case of Figure 4.20, CVIHIS-RTM slightly overestimates its fair share.

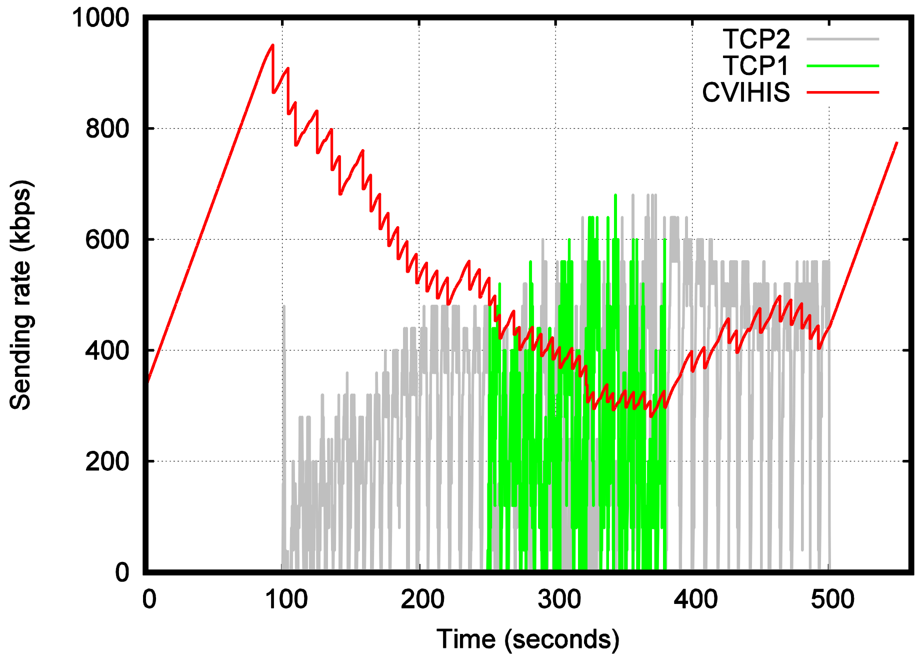


Figure 4.19 Case 1: CVIHIS-RTM against two TCP connections

In the third case, there were two CVIHIS-RTM connections competing against three TCP connections. In this case, the capacity of the bottleneck link was 1.5 Mbps. The queue size of the bottleneck node was 30 packets and the one-way propagation delay was 90 ms. The simulation results of this case are presented in Figure 4.21. Only the sending rates of the CVIHIS-RTM connections are presented in this figure. One of the CVIHIS-RTM connections was sending all the time while the other was sending between 200 and 700 seconds. The number of active connections is presented at the bottom of the figure. CVIHIS-RTM managed in an acceptable manner in this simulation, although CVIHIS-RTM somewhat underestimates its fair share when there is more than one TCP connection in the active state.

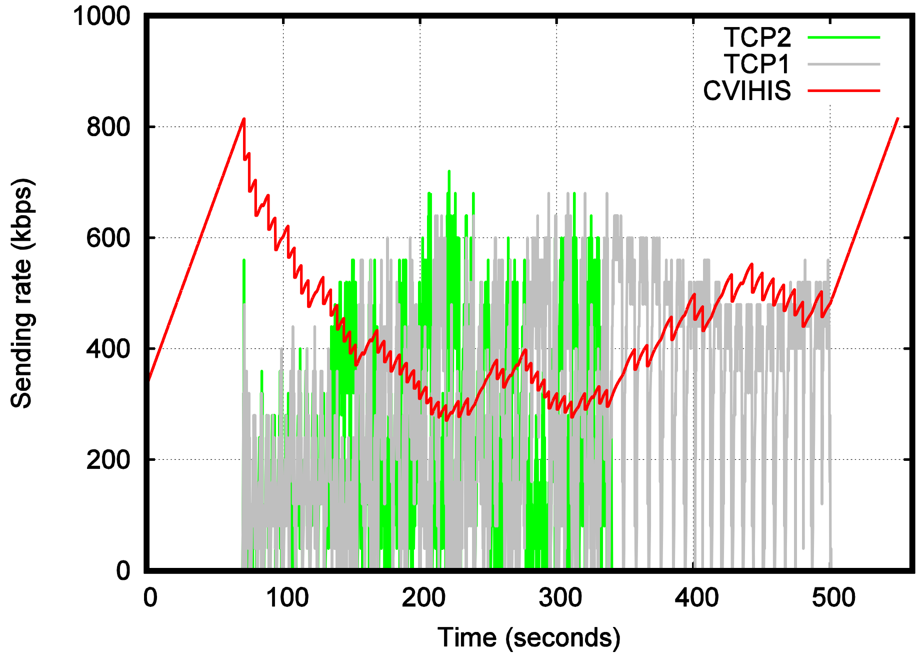
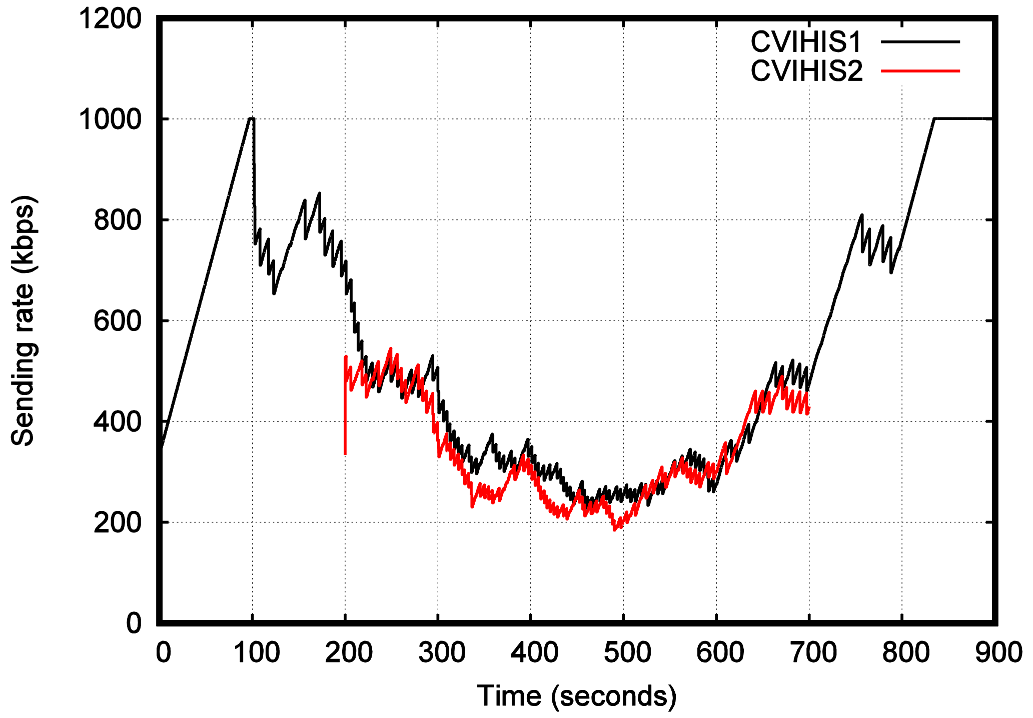


Figure 4.20 Case 2: CVIHIS-RTM against two TCP connections



Number of connections										
	1	1	2	2	2	2	2	1	1	CVIHIS
	0	1	1	2	3	2	1	1	0	TCP

Figure 4.21 Simulation case including two CVIHIS-RTM and three TCP connections

4.2.6.4 Two cases: RED and TCP NewReno

Two simulation cases are presented in this section. First, CVIHIS-RTM was tested against one TCP connection using the RED active queue management mechanism in the bottleneck router. After that, a simulation is presented in which CVIHIS-RTM competed against TCP's NewReno version. These simulations are presented in the same section because, in both cases, it is to be expected that there will be fewer clustering types of TCP packet drops than in the previous simulation cases. It can be assumed that TCP will manage a little better than before because there are now fewer TCP slow starts. These two simulations imitate the simulation case presented in Figure 4.13 as much as possible so that comparisons can be made. Please note that the CVIHIS-RTM sending rate was 307 kbps and that of TCP was 273 kbps in that simulation.

In the RED case, the tail-drop policy is replaced by the RED mechanism. The size of the bottleneck queue was 100 packets because RED drops packets considerably before the queue is full. Four parameters have to be defined for RED. These parameters are presented in section 2.2.2. The minimum threshold parameter *min_th* is 20 packets and the maximum threshold parameter *max_th* is 40 packets. The maximum dropping probability *Pmax* is 0.1 and the smoothing factor *w* for computing the averaged queue length is 0.02. Based on several RED-based simulations, it was found that CVIHIS-RTM also needs tailored RED parameters. In the typical RED parameter group, the smoothing factor w is 0.002 and *max_th* is three times *min_th*. With these typical parameters, the packet dropping level of the queue fluctuates a lot. CVIHIS-RTM does not work properly with these typical parameters because the CVIHIS-RTM smoothness suffers due to the fluctuation of the maximum delay value.

The simulation result of this case is depicted in Figure 4.22. The capacity of the bottleneck link was 600 kbps. Both connections started at the beginning of the simulation and were sending continuously. If the results are compared with the case of Figure 4.13, some differences can be noticed. As expected, the TCP connection here performed better. The sending rate of CVIHIS-RTM was 279 kbps and that of TCP 315 kbps. The rate of the TCP connection also varied in a somewhat smoother manner. The NS-2 simulator shows that there is no clustering type of packet dropping situations any more thanks to the use of RED. The sending rate of CVIHIS-RTM varied more than in Figure 4.13. When RED is used, packets are dropped in a more random way. Due to this randomness, the maximum delay value of CVIHIS-RTM is no longer static and the sending rate of CVIHIS-RTM varies more than in Figure 4.13. In spite of this variation in the sending rate, it can be stated that CVIHIS-RTM performed in an acceptable way in this simulation case.

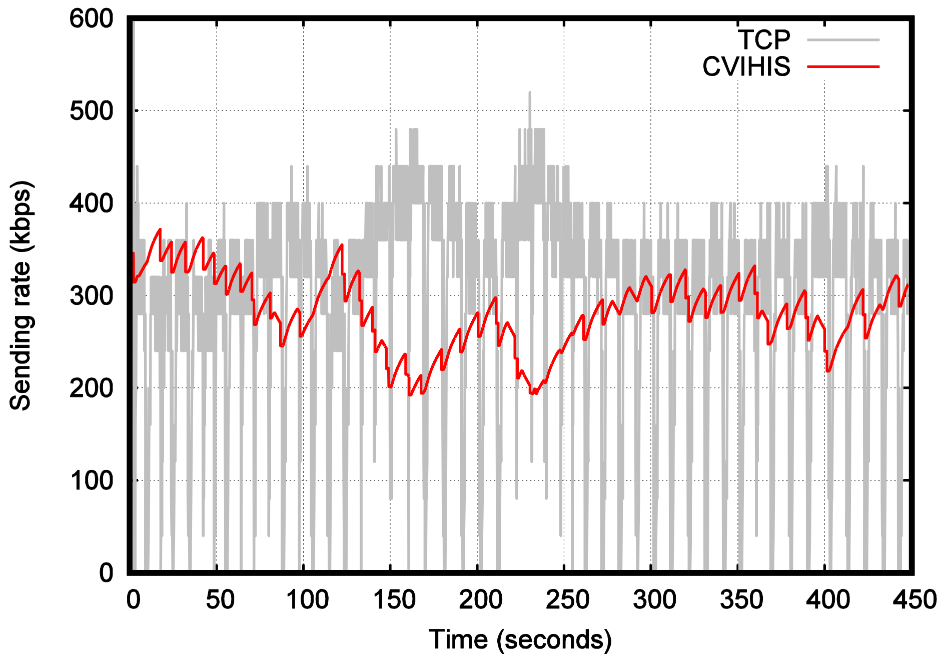


Figure 4.22 Testing CVIHIS-RTM against TCP using RED active queue management

The simulation result of the TCP NewReno case is presented in Figure 4.23. The capacity of the bottleneck link was still 600 kbps and both connections were sending continuously. TCP performed better compared to the results of Figure 4.13 in this case also. The CVIHIS-RTM sending rate was 277 kbps and that of TCP 321 kbps. The rate of the CVIHIS-RTM connection varied in a much smoother manner than in the RED case because the maximum delay value was static. CVIHIS-RTM was also simulated against TCP NewReno using the RED mechanism in the bottleneck router. In this case, the CVIHIS average sending rate was 254 kbps and that of TCP was 341 kbps. The simulation cases of this section show why TCP friendliness is a challenge in today's heterogeneous Internet environment. The CVIHIS parameters can be tuned so that CVIHIS-RTM can manage better, for example, against TCP NewReno. Unfortunately, this probably means that CVIHIS-RTM will manage worse in some other environments.

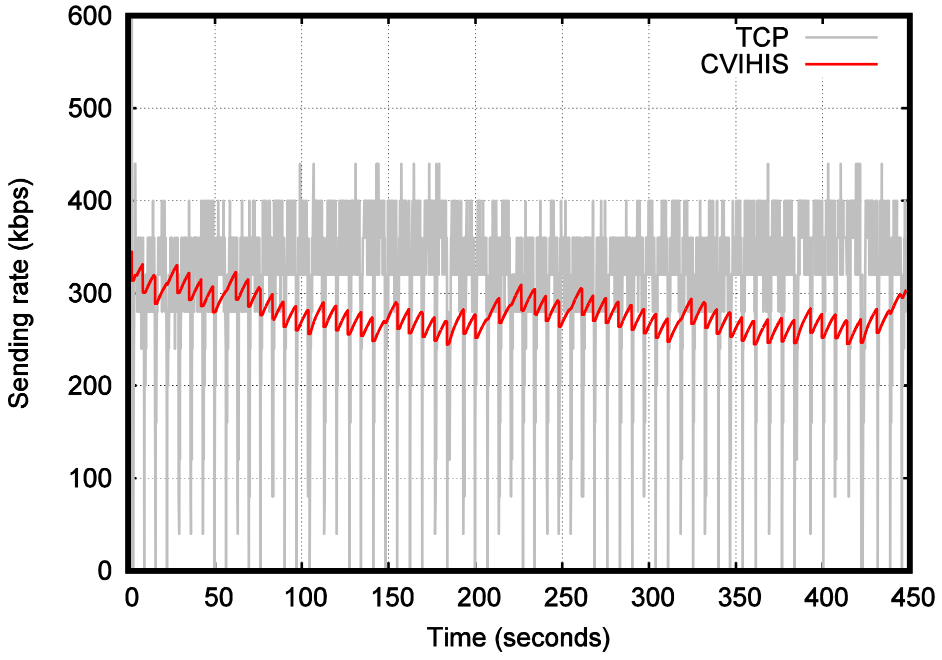


Figure 4.23 CVIHIS-RTM against TCP NewReno connection

4.2.6.5 Case of two queues

The network structures were quite simple in the previous simulation cases. As Figure 4.5 presents, there was only one core link between the end nodes. This also means that there was only one non-empty queue on the connection path. As far as this single non-empty queue condition is met, the behavior of the simple network structure is compatible with that of more complicated network structures. The location of this non-empty queue can change if the size of the queue and the bandwidth of the out-going link remain similar. These kinds of more complicated network structures were taken into account by using different one-way propagation delay values in the previous simulation cases. However, in real networks, it may happen that there are several non-empty queues on the connection path at a certain moment. In this section, the simulation of a case in which there are two non-empty queues on the connection path is described. The case of two non-empty queues affects the maximum delay value in particular, so that it is no longer static. The aim is that the TCP friendliness of CVIHIS-RTM is also at an acceptable level with two non-empty queues on the connection path.

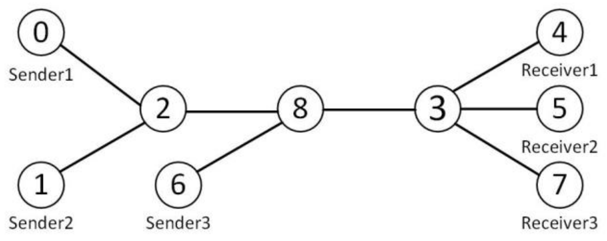


Figure 4.24 Structure of the test network for the case of two queues

The structure of the test network for this case is presented in Figure 4.24. There are three end nodes that send traffic. In all the simulation cases, we are interested in the sending rates of nodes 0 and 1, because these connections have two bottleneck links. Node 0 is the sending node for the CVIHIS-RTM connection and node 1 is the sending node for the TCP Reno connection. These connections send continuously. The third sending node is node 6, sending between 50 and 350 seconds. The sending rate of this node is not directly comparable with that of nodes 1 and 2, because this connection has only one bottleneck link. It also has shorter one-way propagation delay values than the two other connections. It can be expected that node 6 receives somewhat more bandwidth than the other nodes. Nodes 4, 5, and 7 are receiving nodes. On the sender side (nodes 0, 1, 6), the capacity of the access link is 1 Mbps. On the receiver side (nodes 4, 5, 7), this capacity is 1.5 Mbps, so that there is no bottleneck link at the end part of the connection path.

There are now two bottleneck links between the core nodes 2, 8, and 3, because the aim was to test a case in which there are occasionally two non-empty queues on the connection path. These queues are at nodes 2 and 8. The size of these queues varies between 30 and 50 packets. The capacity of the bottleneck link 2-8 is a simulation specific value and varies from 600 kbps to 1.0 Mbps. The capacity of the bottleneck link 8-3 is based on link 2-8. If link 8-3 is too slow, there will be a queue only at node 8. If it is too fast, there will be a queue only at node 2. Therefore, the capacity of link 8-3 should be 1.5 times as much as the capacity of link 2-8, or a little more. This is because link 2-8 has two connections and link 8-3 three connections.

Twelve simulations were made to study the TCP fairness of CVIHIS-RTM. The results of these simulations are presented in Table 4.6. The second and third columns present the capacity of the bottleneck links. In the fourth column, the size of the queues for nodes 2 and 8 is presented. The fifth column presents the one-way propagation delay values used in these simulation cases for connections 0-4 and 1-5. The actual simulation results are presented in the last three columns. Two of these columns present the sending rates of the two connections of interest. Based on the averaged sending rates, the sending rates indicate a good level of fairness. However, the average of Jain’s fairness indices is 0.988, indicating that there are differences in the sending rates. In the worst cases, the faster connection receives about 1.5 times as much bandwidth as the slower connection. The results are also compatible with the previous results because there is a trend that CVIHIS-RTM manages better when the one-way propagation delay values are higher.

The seventh column of Table 4.6 presents the standard deviation of the CVIHIS-RTM sending rate. These results indicate that the sending rate of CVIHIS-RTM varies more in the case of two queues than in the case of one queue. This is because the maximum delay value related to the packet drop situations is no longer static. This delay value varies according to the level of the non-full queue. A similar variation related to the maximum delay value was present in the simulation case of the RED active queue management mechanism (see section 4.2.6.4). The RED simulation case also indicated that the CVIHIS-RTM sending rate varies somewhat more when the maximum delay value varies.

Table 4.6 Simulations for testing CVIHIS-RTM TCP friendliness with two queues

	Capacity of link 2-8 [kbps]	Capacity of link 8-3 [kbps]	Queue size [packets]	One-way delay [ms]	CVIHIS [kbps]	CVIHIS deviation [kbps]	TCP [kbps]
1	600	900	30	120	313	35	245
2	600	1000	40	160	326	26	255
3	600	1100	30	120	271	39	282
4	600	1200	40	80	251	53	294
5	800	1200	30	120	363	37	332
6	800	1200	30	200	380	32	371
7	800	1300	40	160	448	31	329
8	800	1400	40	80	378	41	382
9	1000	1500	40	80	345	60	526
10	1000	1500	30	200	487	56	380
11	1000	1600	50	120	416	56	515
12	1000	1700	40	160	424	65	520
Average					366		368

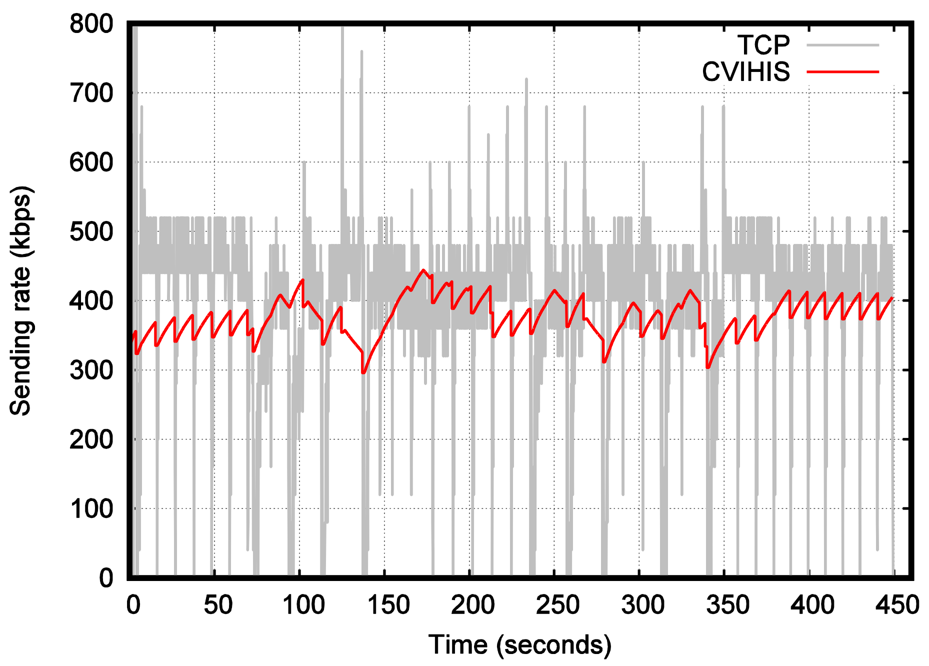
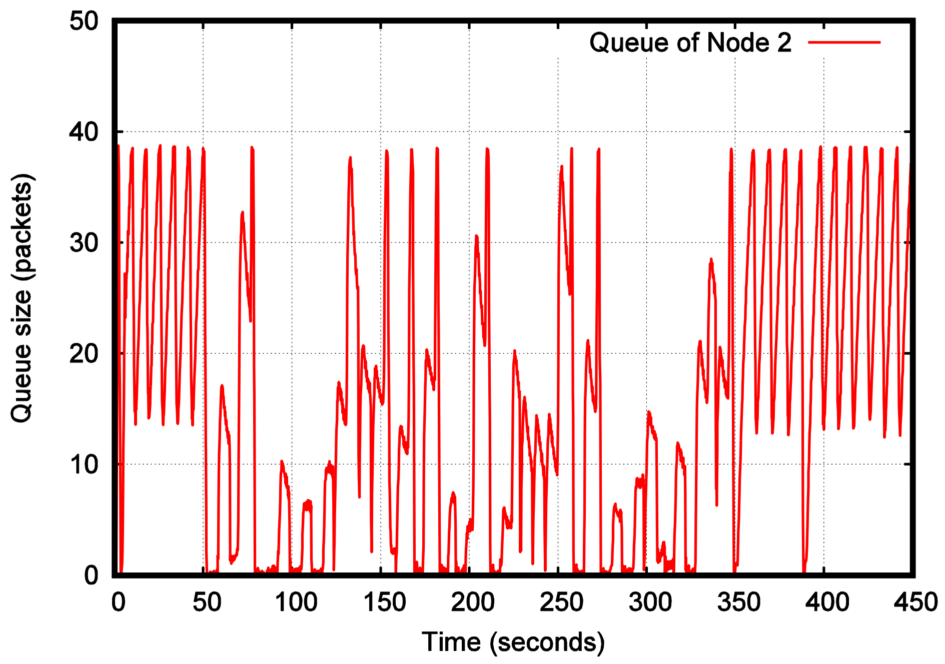


Figure 4.25 Simulation case considering two bottleneck links

The CVIHIS-RTM sending rate fluctuation can also be seen in Figure 4.25. This figure is a graphical illustration of simulation case 8 in Table 4.6. The sending rate varies more in the middle phase of the simulation when there are three connections and two bottleneck links in the active state. The queue behavior of nodes 2 and 8 is presented in Figure 4.26.

In the middle phase of the simulation case, there is often a situation of two non-empty queues.



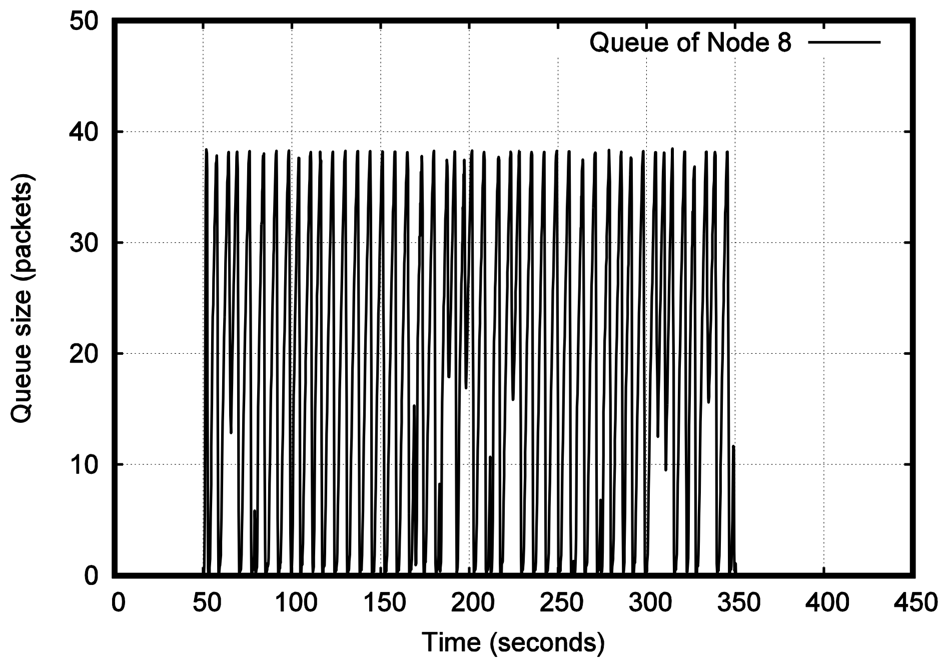


Figure 4.26 Queue behavior in the case of two bottleneck links

4.3. Real network tests of CVIHIS

Simulations are very useful due to their simplicity and flexibility. Unfortunately, simulations do not always adequately reflect real-world situations. Jansen and McGregor (2006) state that simulation tools may use simplified models of TCP, which do not always correspond to real-world implementations in all situations. The paper by Floyd and Kohler (2003) argues that scenarios used in simulations often include certain assumptions. There are simulations with only one congested link. The flows of the congested link share a small range of round-trip times. Most data traffic across the link is one-way and reverse-path traffic is rarely congested. All these assumptions affect the simulation results and the evaluations based on them. Some divergences from reality are unimportant provided that they do not affect the validity of the simulation results. However, it can be difficult to recognise which aspects affect the system behavior in a fundamental way and which aspects can be safely ignored. Related to CVIHIS, the clocks of the end nodes were synchronized in the simulation environment while, in the real network environment, the behavior of CVIHIS can be studied without the synchronization of the clocks. Due to previous issues, it was necessary to test CVIHIS in real network environments. This section presents the results of the real network tests for CVIHIS. Some of the work presented here has been published in Vihervaara et al. (2016b).

4.3.1. CVIHIS implementation for real network tests

In order to test CVIHIS in a real network environment, the CVIHIS code implementation had to be placed somewhere in the protocol stack. There are several possibilities for this placement. For example, an open source operating system could be utilized so that its kernel implementation is modified. In this way, the UDP implementation of the operating system could be changed to correspond to the CVIHIS algorithm. Rather than using such an elaborate although efficient solution, the simplest implementation option was chosen in this thesis. CVIHIS was implemented as a normal socket program on top of the UDP protocol. This solution was possible because the UDP protocol does not offer any special transport services that could disturb the operation of CVIHIS. This option also offers some advantages. UDP port numbers can be used in such a way that the same computer can run a few traffic sources at the same time. This reduces the number of computers needed for the network test. In addition, if large-scale real network testing is done in the future, transferring CVIHIS communication inside the UDP protocol will provide resistance against firewall blocking.

The natural choice is to use the C programming language for this implementation, because it allows us to take advantage of the CVIHIS implementation for the NS-2 simulation program. The receiving-end solutions for the real-environment implementation are fairly similar to those created for the simulation cases. However, the transmitting-end solutions deviate more from each other because the real network version cannot utilize easy-to-use timers and event handlers offered by the simulation environment.

A clock resolution of 0.1 milliseconds was used in the real network tests. This is a more than adequate resolution because the maximum queuing delay is often of the order of several tens of milliseconds. For example, Cisco adheres to the principle that the default maximum queue size of a router corresponds to a queuing delay of 50 milliseconds. The maximum queuing delay was typically between 40 and 60 milliseconds in the real network tests of this study. A lower clock resolution than the one used in this present study is justified with techniques such as CoDel.

4.3.2. Structure of the test network

The typical structure of the test network is presented in Figure 4.27. It comprises four end nodes and two routers. The link between the routers is the bottleneck link. With this simple test network structure, and with the help of the tc program, we can easily emulate different types of network configurations. Tc (traffic control; TC 2016) is the user-space utility program used to configure the Linux kernel packet scheduler.

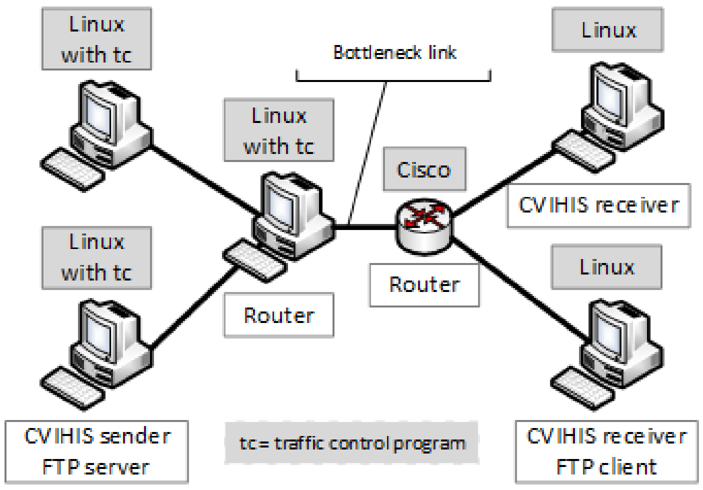


Figure 4.27 Structure of the test network

In the tests relating to this thesis, tc was used in two ways. Tc was used for traffic control in the Linux router. In this way, the capacity of the bottleneck link and the transmission queue size of this link could be varied. When traffic is controlled, the transmission rate of the link is under control. Typically, this means that the available bandwidth is lowered. Traffic control can also be used to smooth the burstiness of incoming links by defining the queue size of the outgoing link. If the queue size is exceeded, the incoming packets are dropped. In the traffic source nodes, tc was used to define the delay characteristics of the outgoing links. In this way, it was possible to emulate different round-trip times of connection paths.

4.3.3. Test results

This section presents the results of the real network test. The aim was that the objectives set for CVIHIS in the simulation tests would also be met in the real network tests. The test results should also be consistent with the results of the simulation tests. In particular, CVIHIS' behavior against itself was considered, because in this respect, the simulation tests were at a preliminary level. In these real network tests, the packet size was 1526 bytes, which is the maximum packet size of the wired Ethernet.

4.3.3.1 Backward-loading mode

The backward-loading mode has to reach the constant sending rate if there is no need for a back off operation. This means that the sending rate should not oscillate. This was ensured by performing twenty tests in the test network. In these test cases, the queue size of the bottleneck link varied between 40 and 60 packets, the capacity of the bottleneck link varied between 2 and 4.5 Mbps, and the round-trip time varied between 10 and 250 milliseconds. The sending rate stabilized in all of the cases.

The second objective for this mode is appropriate back off behavior. It should withdraw from using the bandwidth when the bottleneck is over-utilized for a sufficient amount of time. It should also take advantage of the free capacity released from other connections The back off behavior was tested against one TCP NewReno connection by using the same kinds of test setups as in the case of the stability check. Proper back off behavior was observed in all cases. This was no surprise because the NewReno version of TCP is more aggressive than the Reno version used in the simulation cases.

Figure 4.28 presents the sending rates of the CVIHIS-BLM connections in the tests in which the back off behavior was tested using different round-trip times (10, 100, 200 ms) for CVIHIS-BLM. TCP used a round-trip time of 200 milliseconds in these tests because TCP is less aggressive with long round-trip times. The capacity of the bottleneck link was 3 Mbps. The TCP connections were active between the test time of about 50 and 170 seconds. CVIHIS-BLM can particularly increase its sending rate faster if the round-trip times are short.

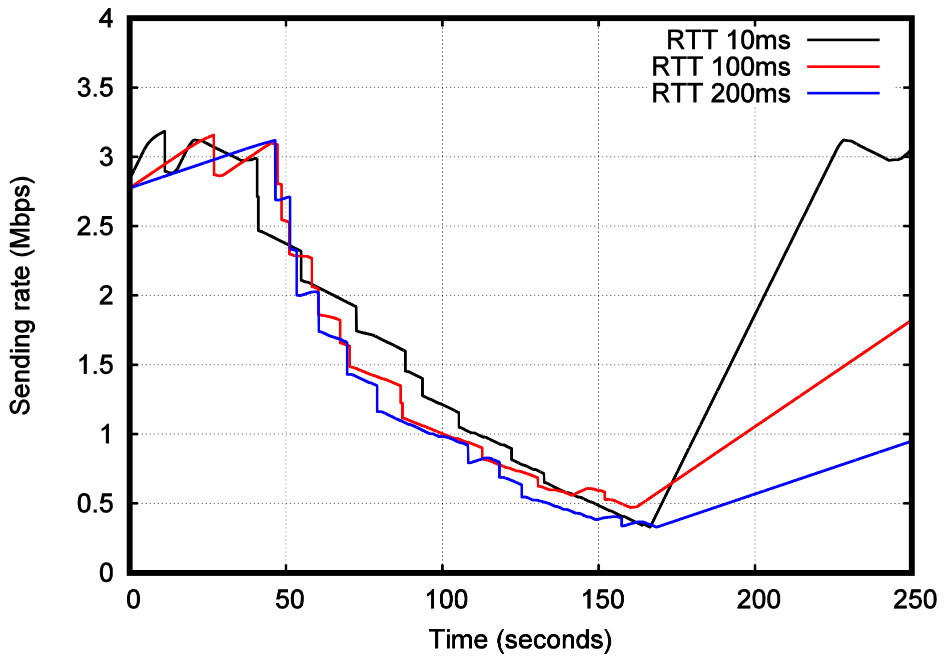


Figure 4.28 Back off behavior of CVIHIS-BLM against one TCP connection

4.3.3.2 TCP friendliness of the real-time mode

The TCP friendliness of the CVIHIS real-time mode was tested against one TCP connection by performing twenty-six tests. The queue size of the bottleneck link was 50 packets. The capacity of the bottleneck link varied between 2 and 4.5 Mbps. Four different round-trip times (20, 80, 140, 200 milliseconds) were used. The starting rates of the connections also varied between the simulations.

These test results are presented in Figure 4.29. As can be seen, individual measurements depart from the trend due to the phase effect. The figure presents the proportion of the CVIHIS-RTM connection from the capacity of the bottleneck link. This figure indicates an acceptable level of averaged fairness. The average of Jain's fairness indices is 0.982, indicating that there are differences in the sending rates. In the worst cases, the connection of higher bandwidth receives about 1.6 times as much bandwidth as the slower connection.

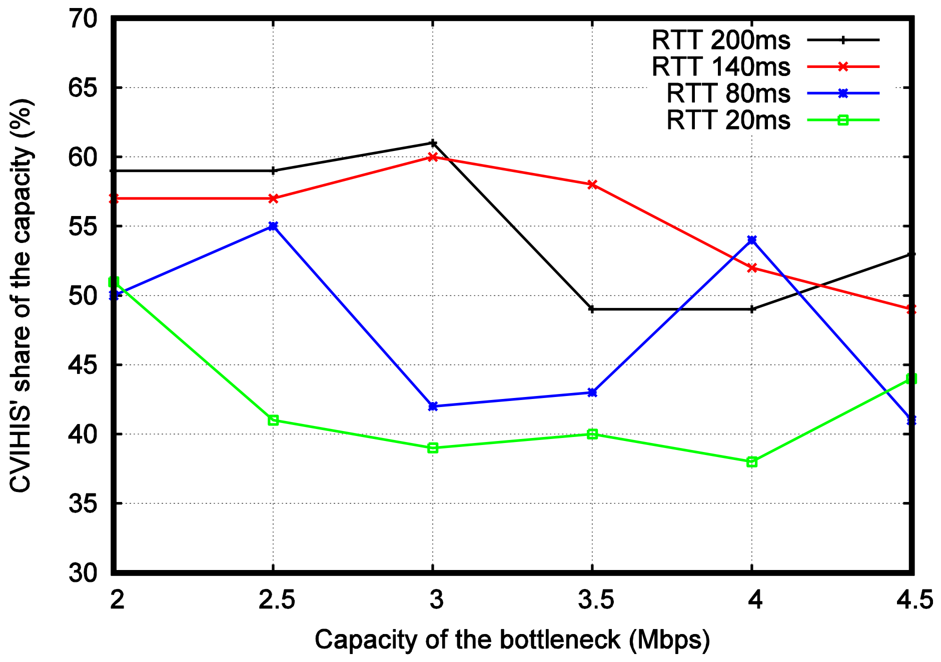


Figure 4.29 Real-time mode against one TCP connection

As mentioned earlier, TCP favors the connections of short round-trip times while CVIHIS-RTM does this in a more modest way. The above results confirm this. Round-trip times affect CVIHIS-RTM less than TCP. CVIHIS-RTM manages relatively modestly when round-trip times are short. When round-trip times are long, CVIHIS-RTM manages somewhat better than TCP.

The Linux version used in the real network environment also supported another TCP congestion control mechanism. This version was CUBIC TCP (Ha et al., 2008). In fact, CUBIC is the current default TCP algorithm of Linux. Some tests were made in which CVIHIS-RTM was tested against the CUBIC version. The preliminary results show that CUBIC behaves somewhat more aggressively than NewReno. If the aim is for CVIHIS-RTM to perform in a friendly way towards the CUBIC version, the rate adjustment parameters of CVIHIS would have to be adjusted slightly so that CVIHIS-RTM behaves even more aggressively.

4.3.3.3 CVIHIS against itself

This section presents the results of tests in which CVIHIS was tested against itself. The results of this thesis show that it is very difficult to achieve an acceptable level of fairness in heterogeneous network environments. It is important that CVIHIS behaves against itself in a fair manner because a workable solution for network congestion control might require that there are only a few different kinds of congestion control mechanisms on the Internet.

The real-time mode of CVIHS was tested against itself by doing 30 tests. In these cases, the queue size of the bottleneck link varied between 40 and 60 packets, the capacity of the bottleneck link varied between 2 and 6 Mbps, and the round-trip time varied between 10 and 240 milliseconds. Tests were also made using different round-trip times for the connections. The starting rates of the connections also varied between the tests. Based on these tests, the sending rates indicated a good level of fairness. In most cases, transmission rates differed less than 10 percent. Only when round-trip times were substantially different were the rate differences larger than 10 percent. When the worst cases are examined, the connection of higher bandwidth received about 1.7 times as much bandwidth as the slower connection. In this case, the faster and slower connections had round-trip times of 10 and 180 milliseconds, respectively.

One of these tests is presented in Figure 4.30. The capacity of the bottleneck link was 4 Mbps. The round-trip times of the CVIHIS-RTM connections were 60 (red) and 180 (black) milliseconds. In this case, the average sending rates were 2.085 Mbps and 1.935 Mbps. The first 50 seconds were omitted when calculating the averaged rates.

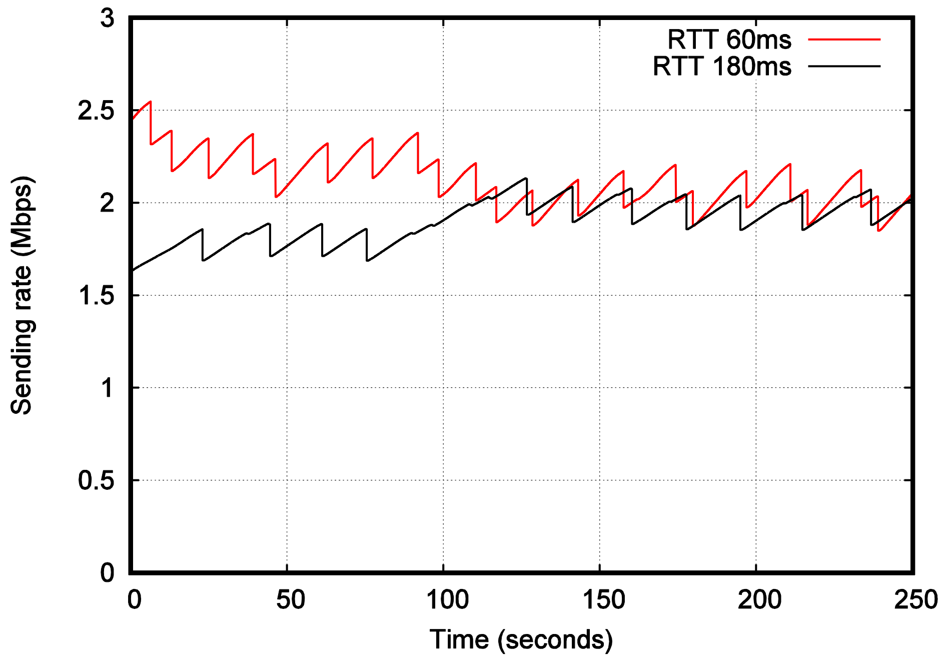


Figure 4.30 CVIHIS-RTM against itself

Some tests were made so that there were four active CVIHIS-RTM connections in the network at the same time. CVIHIS-RTM performed acceptably in these tests although it took more time to balance the sending rates in the cases of long round-trip times. The result of one such test is presented in Figure 4.31. In this case, the capacity of the bottleneck link was 10 Mbps and the round-trip time was 30 milliseconds. The queue size of the bottleneck link was 60 packets. The number of active connections is presented at the bottom of this figure.

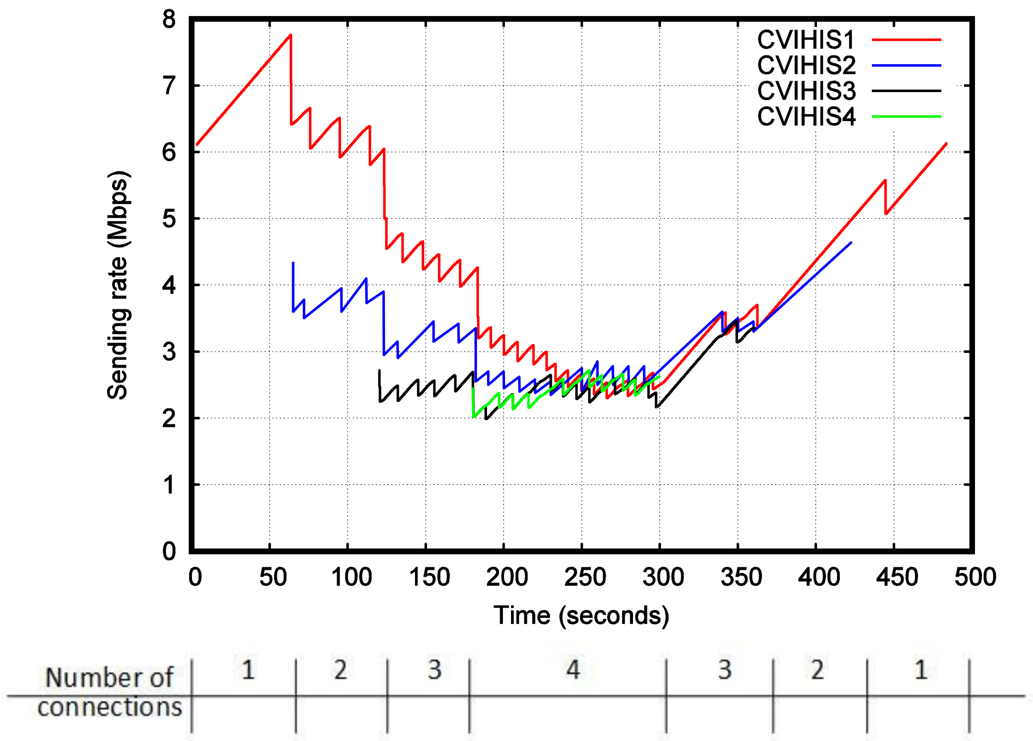


Figure 4.31 Four real-time mode connections

The back off behavior of CVIHIS-BLM was also tested when there was a CVIHIS-RTM connection on the connection path at the same time. 50 tests were made so that the capacity of the bottleneck link was 2 or 4 Mbps. The queue size of the bottleneck link was 60 packets. The round-trip times varied from 10 to 200 milliseconds.

When both modes used the same round-trip time, the back off behavior was as expected. The tests were also made so that the modes used different round-trip times. In these cases, the back off actions happened slowly if the round-trip time of the real-time mode was much longer than the round-trip time of the backward-loading mode. If the round-trip times differed considerably, for example, ten times, the back off action did not take place at all. This behavior is presented in Figure 4.32. The figure shows the sending rates of CVIHIS-BLM in three different cases. The capacity of the bottleneck link was 2 Mbps. The round-trip times of CVIHIS-RTM were 10, 150, and 200 milliseconds. The real-time connections were active from the test time of 20-40 seconds to the test time of 200-220 seconds. In all these cases, the round-trip time of CVIHIS-BLM was 50 milliseconds. It is easy to moderate this phenomenon. The backward-loading mode could use less aggressive rate adjustment parameters than the real-time mode. In this thesis, both modes used the same parameter set.

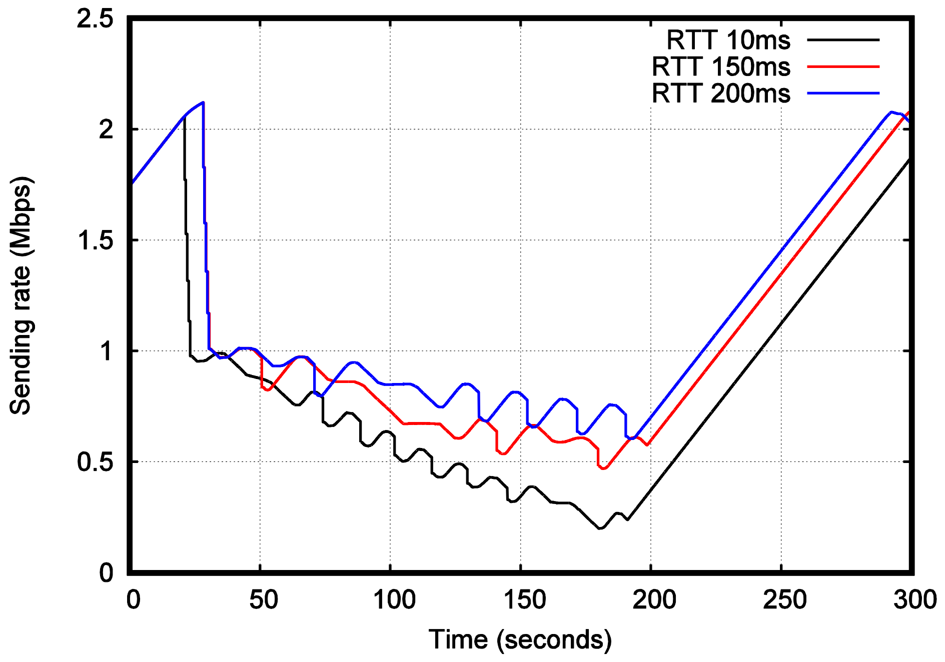


Figure 4.32 Back off behavior of CVIHIS-BLM against CVIHIS-RTM

4.4. Parameter sensitivity of CVIHIS

There are six parameters to adjust CVIHIS' behavior. These parameters have already been presented in Tables 4.1 and 4.2. In addition to finding suitable parameter groups, observations were also made about the sensitivity of the parameters. It was found that CVIHIS behaves in quite a parameter-insensitive way. A small change in the values of the adjusting parameters caused only small changes in the test results. For example, regarding TCP friendliness, if a suitable parameter group was tuned in a minor way to favor the other protocol, this other protocol still did not dominate.

Table 4.7 presents a short overview on the parameter sensitivity of the CVIHIS real-time mode related to TCP friendliness in the simulation cases. The first and second rows are the same as in Table 4.1. The third and fourth rows present the actual simulation results with respect to parameter sensitivity. The parameter values were changed one at a time so that the other parameters used the values of the suitable parameter group presented in the second row. The first column is not the real *tuning* parameter because a constant factor of 1.0 was used for this parameter. The cells of the third and fourth rows present the actual simulation results (i.e., the average sending rate of CVIHIS and, in round brackets, the standard deviation of the sending rate). The simulation case is otherwise similar to that of Figure 4.13. In this simulation case, the average sending rate of CVIHIS-RTM was 307 kbps and the standard deviation of the CVIHIS-RTM sending rate was 20 kbps.

Table 4.7 Parameter sensitivity of CVIHIS-RTM

--- (1) +++ (4)	-- (2)	++ (3)	- (6) + (7)	MD (5)	SF	PF
1.0	0.7	0.5	0.2	1.25	0.4*last update 0.6*history	1.02
-	0.75 296 kbps (19 kbps)	0.55 307 kbps (18kbps)	0.25 304 kbps (17 kbps)	1.30 273 kbps (25 kbps)	0.35*last update 0.65*history 309 kbps (24 kbps)	1.03 299 kbps (18 kbps)
-	0.65 306 kbps (24 kbps)	0.45 299 kbps (19 kbps)	0.15 307 kbps (18 kbps)	1.20 323 kbps (17 kbps)	0.45*last update 0.55*history 317 kbps (19 kbps)	1.01 311 kbps (18 kbps)

The majority of the parameters caused only small changes to the sending rates. The multiplicative decrease factor (MD) presented in the fifth column caused the most significant changes in the sending rates. The right value of the smoothing factor parameter (SF) proved to be important as well. In the suitable parameter group, the history of the sending gap values was weighted by a factor of 0.6, and the newly calculated sending gap by a factor of 0.4. Moderate changes in these parameter values did not cause any significant changes in sending rates, However, if the factor related to the last update was shifted close to a value of 0.3, CVIHIS reacted too slowly to changes in the network. As a result, the TCP friendliness of the real-time mode suffered or the backward-loading mode could not reach the targetDelay area. If the last update factor approached a value of 0.5, the smoothness of the real-time mode suffered.

4.5. Comparison with competing mechanisms

In this section, CVIHIS is compared with two other congestion control mechanisms. The backward-loading mode is compared with LEDBAT. LEDBAT is designed for low priority applications and it has been used by some background bulk-transfer applications. It has been used by Apple for software updates, and by BitTorrent, for example. The real-time mode of CVIHIS is compared with Google Congestion Control for Real-Time Communication on the World Wide Web (GCC). GCC tries to be TCP-friendly. It has already been implemented in the Google Chrome and Firefox browsers. LEDBAT and GCC have been described in Chapter 3.

4.5.1. Backward-loading mode versus LEDBAT

The LEDBAT comparison was made using the NS-2 simulator. The NS-2 source code for the LEDBAT implementation is provided on the LEDBAT software page. Figures 4.33-4.36 present the results of two simulation cases where the back off behavior of

LEDBAT was tested based on the network structure of Figure 4.5. In these two cases, the capacity of the bottleneck link was 600 kbps and the one-way propagation delay was 90 milliseconds. The queue size of the bottleneck link was 30 packets.

In the first simulation case, the target queuing delay for the LEDBAT connection was 60 milliseconds. Figure 4.33 presents the size of the LEDBAT congestion window for this case. Figure 4.34 presents the queue size of the bottleneck node. In the second simulation case, the target queuing delay for the LEDBAT connection was 55 milliseconds. Figure 4.35 presents the size of the LEDBAT congestion window for this case while Figure 4.36 presents the queue size of the bottleneck node. In both cases, the LEDBAT connection was started at the beginning of the simulation. We waited until the sending rate of LEDBAT settled to the capacity of the bottleneck link. After this, the TCP Reno connection was started. The TCP connection was active between 50 and 150 seconds.

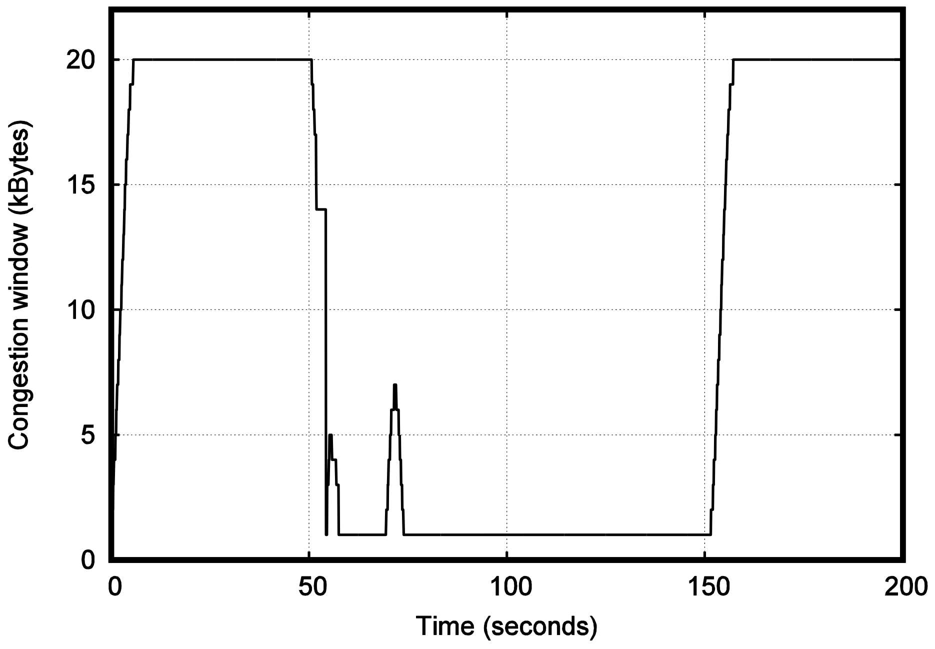


Figure 4.33 LEDBAT congestion window size when the target queuing delay was 60 ms

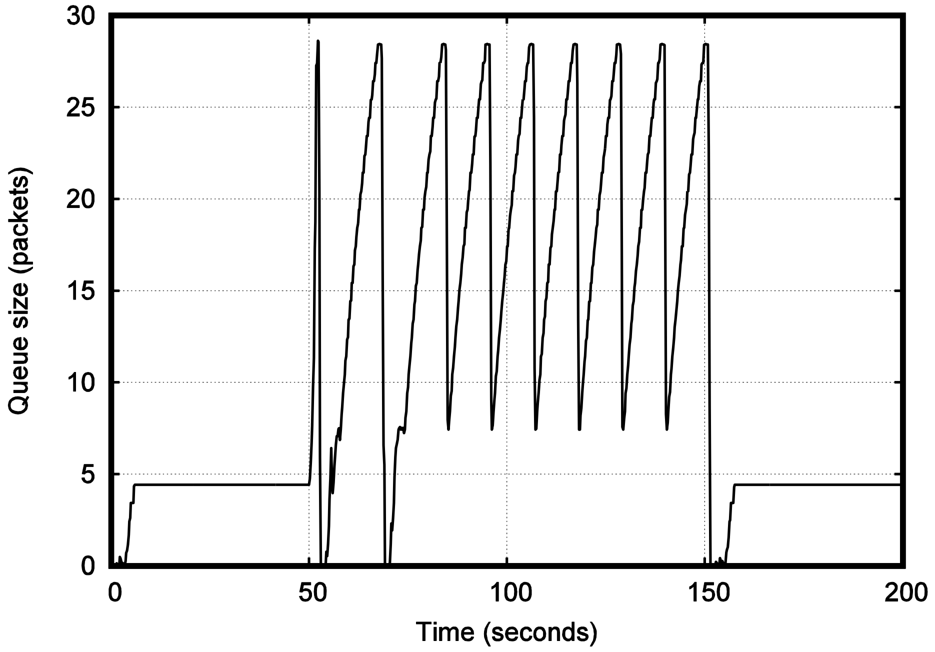


Figure 4.34 Queue behavior when the target queuing delay was 60 ms

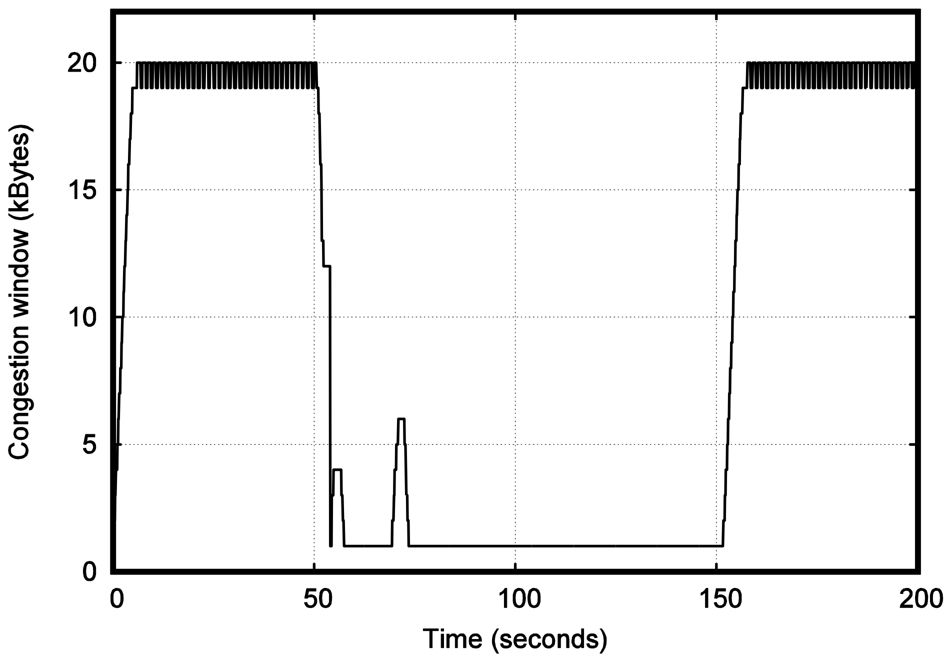


Figure 4.35 LEDBAT congestion window size when the target queuing delay was 55 ms

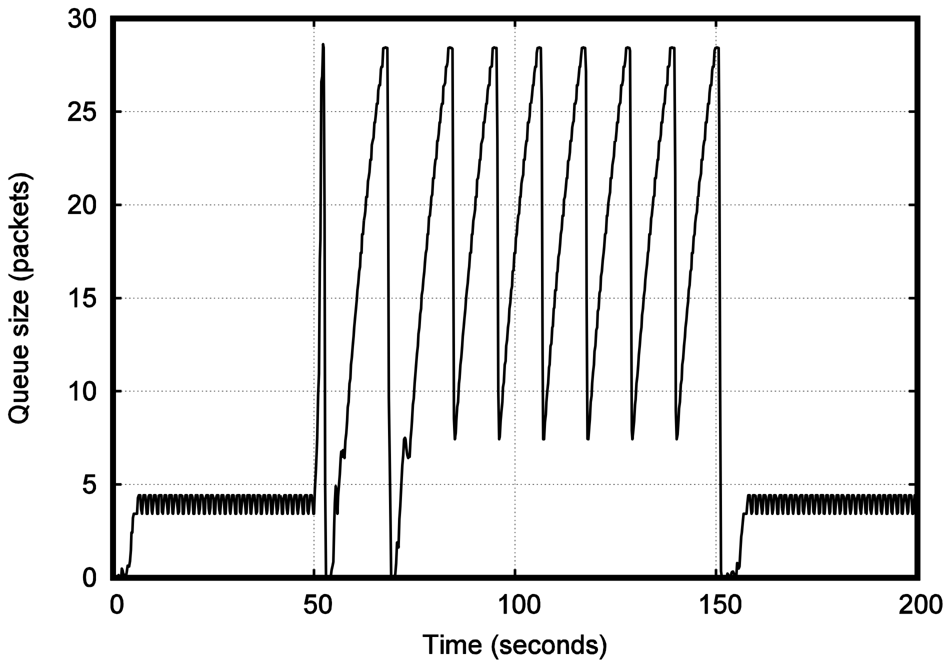


Figure 4.36 Queue behavior when the target queuing delay was 55 ms

These two simulation cases were designed so that the results in Figures 4.33-4.36 could be compared with the figures describing CVIHIS-BLM behavior, presented in sections 4.2.2 and 4.2.3. Both mechanisms have the ability to stabilize their sending rate to the capacity of the bottleneck link. CVIHIS-BLM can do this stabilization without significant oscillation in every case. Instead, as presented in Figure 4.35, the sending rate of LEDBAT oscillated slightly in some cases. The reason for this oscillation is that the LEDBAT target delay is a strict value rather than a range. In addition, the LEDBAT sending rate is controlled in a window-based manner, which is sensitive to the phase effect. In contrast, the CVIHIS-BLM target delay is an area and the CVIHIS sending rate is controlled in a rate-based manner. Because the CVIHIS-BLM target delay is an area, the queue level varies more with CVIHIS-BLM.

As the figures present, LEDBAT can accommodate its sending rate more quickly than CVIHIS-BLM. This behavior has its advantages and disadvantages. If free capacity becomes available at the bottleneck router, LEDBAT can take it quickly into use. On the other hand, if this free capacity is temporary, the sending rate must be reduced radically somewhat later. This kind of situation can be seen in Figures 4.33 and 4.35 at a time unit of 70 seconds. Free capacity becomes temporarily available in the network because the TCP connection makes its slow start after the packet drop. Figure 4.8 shows that CVIHIS-BLM reacts in a more moderate way when there is a TCP slow start.

Generally speaking, CVIHIS-BLM is somewhat more flexible because it can acquire the values of the targetDelay area without manual configuration. CVIHIS-BLM can automatically take into consideration the properties of the connection path. With

LEDBAT, the target delay is configured manually. On the whole, it is difficult to say which mechanism is better because LEDBAT and CVIHIS-BLM work in a slightly different way. However, we can draw the conclusion that CVIHIS-BLM is competitive with LEDBAT.

4.5.2. Real-time mode versus GCC

The GCC comparison was made with the help of Carlucci et al. (2016). In their paper, they experimentally evaluated GCC by setting up a controlled testbed. In this testbed, two hosts were connected directly using a network cable. The network between these hosts was emulated by software. The NetEm Linux module along with the traffic shaper set delays for connection paths and available bandwidths for bottlenecks. This test setup does not depart significantly from our real network test setup. They also used the network capacities that have been used in this thesis. The capacity of the bottleneck varied between 500 and 4000 kbps. They sent video traffic between the hosts. Using the testbed, they evaluated to what extent GCC flows were able to track the available bandwidth, minimize queuing delays, and fairly share the bottleneck with other GCC or TCP flows.

They found that GCC flows were able to track the available bandwidth of the empty network so that GCC used slightly over 80 percent of the bandwidth. If this result is compared with Figure 4.10, it can be seen that CVIHIS-RTM can use almost 100 percent of the bandwidth. On the other hand, GCC has a fairly good ability to minimize queuing delays. The real-time mode of CVIHIS-RTM can minimize queuing delays if congestion notifications are sent when queuing delays are still at a moderate level. This can be done by sending explicit congestion notification well before the queue actually overflows. The number of dropped packets can also be minimized using ECN. Because the sending rates of both mechanisms fluctuate smoothly, they also take the free capacity for their use slowly. The sending rate of GCC varies slightly more than that of CVIHIS-RTM.

Carlucci et al. (2016) found that when three GCC video flows shared the bottleneck, the bandwidth was shared quite fairly. The measured Jain’s fairness index was 0.93. If these results are compared with the results in sections 4.2.5 and 4.3.3.3, it can be seen that CVIHIS-RTM behaves against itself better. In all the cases described in these sections, Jain’s fairness indices are over 0.99 when considering the phases where the sending rates have stabilized. They also found that three GCC flows were able to track the available bandwidth so that about 80 percent of the bandwidth was used. In contrast, CVIHIS-RTM can use almost 100 percent of the bandwidth in a similar case.

The TCP friendliness of GCC was tested against 99 TCP connections. The results showed that TCP friendliness of GCC was at an acceptable level. When the test results of Carlucci et al. (2016) are considered regarding TCP friendliness, it is worth remembering that the results are affected by the TCP version used. They used the TCP CUBIC congestion control, since it is the default version used by the Linux kernel. In TCP friendliness, comparison cannot be made with CVIHIS-RTM because the GCC test results were quite limited. In particular, the results with respect to different RTT times are limited. However, they have performed a broader experimental evaluation than the one presented in Carlucci et al. (2016).

One disadvantage of GCC could be that it increases its sending rate in a multiplicative manner instead of using additive steps. The first version of CVIHIS (Vihervaara and Loula 2014) as well as the work by Chiu and Jain (1989) showed that increasing sending rates in a multiplicative manner could pose challenges for TCP friendliness and friendliness against itself. We can conclude this subsection by stating that CVIHIS-RTM is at the least competitive with GCC. In some cases, CVIHIS-RTM outperforms GCC. The two mechanisms are quite similar but CVIHIS-RTM is able to use bandwidth more efficiently than GCC, whereas GCC is able to control network delays better than CVIHIS-RTM.

4.6. Summary based on the test results of this chapter

Hundreds of tests have been executed to justify the proper operation of CVIHIS. In this section, the results of these tests are summarized. The simulation and real network test results are consistent with each other. The backward-loading mode had two targets for proper operation. The first target was that this mode should be able to reach the constant sending rate based on the capacity of the bottleneck link provided that there is no need for the back off operation. This means that the sending rate should not oscillate constantly. The sending rate stabilized in all the test cases. The queue behavior of the bottleneck link also confirmed the desired behavior. Generally speaking, stabilization also means that if there are several backward-loading mode connections active at the same time, these connections stabilize together to the capacity of the shared bottleneck link. In cases like this, the stabilized rates can differ from each other. Some simulations were made to verify this behavior. Please note that these different rates are not against the proper function of the backward-loading mode because this mode does not have bandwidth and fairness demands.

The second target of the backward-loading mode was that it should back off when there are bandwidth demands by other connections. The back off behavior was tested against TCP connections. Proper back off behavior was observed in all the cases. It was also noticed that the back off operation takes some time because EWMA is used to smooth the sending rate. Based on the tests, it can be concluded that the backward-loading mode behaves as desired and expected.

The main target of the real-time mode was to be TCP-friendly. This means that this mode should compete with TCP for the use of bandwidth in a fair manner. Based on the tests results, the conclusion can be drawn that CVIHIS does not significantly discriminate against connections in respect of heterogeneous round-trip times. This feature has its advantages as well as disadvantages. The TCP friendliness of CVIHIS-RTM suffers when the round-trip times are very short or long. In the mid-delay area, the sending rates are quite equal. On the other hand, CVIHIS can alleviate unfairness related to heterogeneous round-trip times. In addition, CVIHIS-RTM behaves against itself quite fairly in spite of the heterogeneous delay properties of the connections.

When the test results related to TCP fairness are analyzed, it can be observed that the averaged results indicate a good level of fairness. In the worst cases, faster connections receive about 1.7 times as much bandwidth as the slower connections. Unfortunately, it is difficult to define an acceptance level related to TCP fairness. It is partly based on personal opinion because there is no widely accepted definition for TCP fairness. As an example related to DCCP's TFRC (Floyd et al. 2008), TCP friendliness is defined as:

TFRC is designed to be reasonably fair when competing for bandwidth with TCP flows, where we call a flow "reasonably fair" if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions.

When individual results are considered, the main reason for unfairness between competing flows is often that TCP favors connections with shorter round-trip times. Tsao et al. (2007) defines the concept of TCP-equal share as:

A TCP-equal share flow uses the same bandwidth as a TCP flow if both flows compete for the same bottleneck.

Achieving a TCP-equal share is challenging, because competing for the same bottleneck does not require that competing flows should have identical network conditions. For example, the flows may have different round-trip times. If the congestion control mechanism is biased against long round-trip times, it is an impossible task for the mechanism to achieve a TCP-equal share. In contrast, when the average results are considered, it can be stated that CVIHIS-RTM promotes the TCP-equal share concept because CVIHIS is insensitive to heterogeneous round-trip times.

It is also difficult to get different kinds of mechanisms to work together smoothly in different environments. TCP and CVIHIS-RTM are different kinds of congestion control mechanisms. TCP is a loss-based mechanism whereas CVIHIS-RTM is a delay-loss-based mechanism. TCP is a window-based mechanism while CVIHIS is a rate-based mechanism. The paper by Widmer et al. (2001) analyzes various types of congestion control mechanisms. Their paper shows that rate-based congestion control mechanisms often have trouble with TCP friendliness because TCP uses window-based rate control. In contrast, window-based congestion control schemes generally show good TCP friendliness. With this in mind, it can be stated that the TCP fairness of CVIHIS-RTM is at an acceptable level.

It was found that the pushing operation of the minimum delay value is a working solution for delay-based congestion control mechanisms so that these delay-based mechanisms can compete with loss-based congestion control mechanisms fairly. The basic mechanism of CVIHIS works with short-lived connections but the simulation results show that it is more suitable for long-lived connections. As this basic mechanism does not have a special start phase algorithm, it can take time before the congestion control mechanism of CVIHIS really activates after the first packet drop. In addition, the real network tests verified preliminarily that the basic mechanism of CVIHIS works without synchronization of the end node clocks.

4.7. Benefits for current video streaming practices

This section presents how CVIHIS can serve the current video streaming practices better than the existing mechanisms. The analysis is based on the problems with DASH due to the TCP congestion mechanisms described in section 3.4.5. It should also be noted that the backward-loading mode is well suited for situations where videos are downloaded in a push-based approach from primary servers to surrogate servers in content distribution networks.

In DASH, the problems caused by TCP are unfairness and the inefficient use of bandwidth, especially in the cases of heterogeneous or long round-trip times. The test results prove that CVIHIS-RTM behaves very fairly against itself. CVIHIS-RTM can also share bandwidth with TCP traffic sources at an acceptable fairness level. The test results show that CVIHIS efficiently utilizes existing bandwidth. In addition, CVIHIS is biased against long round-trip times in a much more moderate way than TCP.

The main advantage of CVIHIS over DASH is its smooth sending rate. This prevents the on-off traffic pattern that TCP is prone to produce. The on-off traffic pattern exposes the receive buffer to level fluctuations. If the DASH rate adaptation is based on the level of the receive buffer, the level fluctuations exposes to the video encoding rate switches of DASH. The switching amplitudes can also be noticeable because the on-off traffic pattern tends to generate fast and large-scale buffer level changes. The smooth sending rate enables smooth rate switching behavior and also reduces the risk of stalling events because it eliminates off-phases, which can cause the receive buffer to unexpectedly drain. However, the blocking of stalling events is ultimately based on the appropriate receive buffer management practices implemented by the DASH client.

A continuous and smooth sending rate also makes it easy to evaluate the current network bandwidth if the DASH rate adaptation is based on the network bandwidth estimation of the DASH client. In reality, in the case of CVIHIS, the DASH client does not necessarily need to evaluate the current bandwidth. Because CVIHIS is a rate-based congestion control mechanism, the CVIHIS server knows exactly the current fair share of the bandwidth. This information can be passed on to the DASH client using the appropriate protocol mechanisms. The DASH client would be responsible for ensuring that the selected chunk rate is also appropriate for the current receive buffer level and terminal properties. This would promote the correct separation of the protocol layers since only CVIHIS would be responsible for evaluating network throughput capabilities.

5 Solutions for Special Cases

Only the basic congestion control algorithm for CVIHIS was presented in the previous chapter. This chapter deals with refining and elaboration work. Solutions and points of view regarding special congestion control cases are presented in this chapter. Most topics are only discussed but in the case of one topic, a simulation is presented.

5.1. Start phase of the connection

The start phase of a connection can be considered from the connection or network point of view. The connection aspect means that the connection wishes to get its fair share of the network bandwidth soon enough. The network aspect means that it is undesirable to have a situation where a new connection, with an overly aggressive sending policy, drives the network to the congestion state. Therefore, if special start mechanisms are used, it is desirable that both these aspects be taken into consideration.

With the backward-loading mode of CVIHIS, there is not much need for a special start phase algorithm. The connection aspect is automatically satisfied because the connection of CVIHIS-BLM has no bandwidth demands, except for the defined minimum sending rate. A connection can start by using an appropriate initial rate. After the start, the rate increase behavior can be based on the operation of the basic CVIHIS-BLM algorithm. In Table 4.3, it typically takes about 30 seconds to achieve a high enough rate. 30 seconds is not much time compared to the typical lifetime of long video connections. The network aspect is also satisfied by this behavior because CVIHIS includes a similar self-clocking mechanism as TCP. The senders of CVIHIS need acknowledgements to increase their sending rate. The connection cannot increase its sending rate too aggressively. If a new connection enters the network when the network is in the congestion state, the sender will receive acknowledgements at a slow rate. In a very bad situation, the sender will not receive any acknowledgements at all.

On the other hand, the real-time mode has bandwidth demands. It is perhaps appropriate to use some kind of a start phase algorithm with this mode so that CVIHIS-RTM can achieve its fair share of the network bandwidth sooner than the test cases mentioned in Chapter 4. However, it is worth noting that the video connections CVIHIS is designed for are typically long-lived connections and the start phase takes only a few seconds. It is not totally certain whether it is worth a more complicated implementation to equip the CVIHIS-RTM congestion control with a special start algorithm. If some kind of a start

phase algorithm is provided, it should take into account the connection aspect as well as the network aspect.
This type of algorithm can be based on the Packet Pair approach because the available capacity of the bottleneck link can be probed by this approach. This kind of start phase behavior has been presented and analyzed by Man et al. (2006) and Zhang et al. (2012) with good results. An example can be used to illustrate how this technique can improve the start phase of CVIHIS-RTM. Consider the situation where CVIHIS-RTM sends packets to the network using a packet interval gap of 0.2 seconds. Instead of sending all the packets using this equal interval gap, the sending gaps can be varied a little. Two consecutive packets can be sent using an interval gap of 0.3 seconds for the purpose of compensation. This longer gap makes it possible to send two other consecutive packets using a sending gap of 0.1 seconds. These two packets are so-called probe packets and the point of interest is the acknowledgement gap of these two packets. If the ACK gap is also 0.1 seconds, the conclusion can be made that there is enough capacity for a sending gap of 0.1 seconds inside the network. Note that if the probe packets are queued on the connection path, the ACK gap will be longer than 0.1 seconds. In practice, the sending gap is changed so that the new sending gap is set to midway between the old gap and the ACK gap. For example, if the ACK gap is 0.12 seconds, the sending gap is changed from 0.2 seconds to 0.16 seconds.

It is preferable that the rate change decision is based on a couple of probe measurements to compensate for the queuing behavior. This probe gap can be calculated on the receiver side if CVIHIS-RTM wishes to eliminate the backward path behavior from the calculation. The result must be echoed back to the sender and therefore, the header structure becomes more complicated. Figure 5.1 depicts how the sending rate increases with this start algorithm. The rate increase behavior excellently satisfies the connection aspect as well as the network aspect. The sending rate increases by large steps at the beginning of the start phase but the steps decrease considerably when the connection approaches the equilibrium level. TCP's slow start behavior is also presented in the figure. TCP behaves almost in the opposite manner. At first, the sending rate increases slowly but the increase becomes considerably faster near the equilibrium level. It is a known problem of TCP's slow start operation that it can exceed the equilibrium level dramatically. Some simulations described in Chapter 4 also showed this problem. Note that the figure only presents the ways in which the sending rates are increased. The figure does not answer the question as to which algorithm reaches the equilibrium level first.

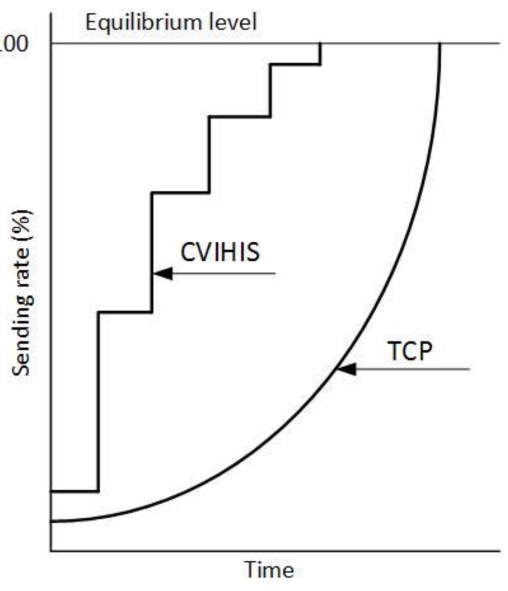


Figure 5.1 Start phase behavior of CVIHIS-RTM and TCP connections

The described start phase algorithm can also be useful in a situation where the sending mode is changed from the backward-loading mode to the real-time mode because the backward-loading mode can operate at a slow sending rate. If the mode change is made in the opposite direction, there is no need for a special algorithm. In this situation, the connection can continue with its fair share rate. After the mode change, the connection has to start to probe if it has to give up bandwidth.

5.2. Non-congestion losses

There were no non-congestion packet losses present in the simulations made in Chapter 4. In those simulation cases, all packet losses were generated by queue overflows. The test network used for real network tests was also not prone to non-congestion packet losses because it used the wired Ethernet. In real networks, packet losses also occur for reasons other than congestion. For example, packets may be dropped by routers or sender side protocol stacks, due to bit errors. This must be taken into consideration by CVIHIS because the maximum delay value is predisposed to non-congestion losses because this value is changed after every packet loss. If packets are dropped due to non-congestion losses, inaccurate information is used for congestion control.

There are at least three ways to alleviate this problem. Firstly, an Exponentially Weighted Moving Average (EWMA) type algorithm can be used for the calculation of the maximum delay value. This EWMA mechanism does not totally neglect non-congestion losses, but it uses an averaging function so that congestion losses and non-congestion losses are averaged. Using EWMA to average the maximum delay value has another benefit as well. There may be different kinds of congestion situations in a network, which can generate different kinds of delay values for packet drop situations. In some cases, there may be only one congested router on the connection path. On the other hand, there may be cases of one congested router and another router with incipient congestion. EWMA makes it possible to average different types of congestion situations.

Another way to manage non-congestion losses is to use a similar kind of approach to the TCP Veno version (Fu and Liew 2003). If the delay value related to a packet loss is near the minimum delay value, the loss is probably a non-congestion loss. If the delay value is near the maximum delay value, the loss is probably a congestion loss. The EWMA and TCP Veno mechanisms could also be combined. The maximum delay value ($maxDelay$) could be updated after a packet loss:

$$k = \frac{lastDelay - minDelay}{\mathrm{maxDelay(n)} - \mathrm{minDelay}} \tag{5-1}$$

$$maxDelay(n+1) = (1 - k * p) * maxDelay(n) + (k * p) * lastDelay \tag{5-2}$$

where:
k = the factor evaluating the probability that a packet loss is caused by a congestion
$lastDelay$ = the last measured one-way propagation delay value
$minDelay$ = the minimum delay value
maxDelay = the maximum delay value
p = the smoothing factor for EWMA.

The best solution for non-congestion losses would be the use of explicit congestion notification (ECN). Since a drop equivalent situation is explicitly indicated by ECN, CVIHIS can be sure that there is a congestion situation in the network. CVIHIS can adopt a rule by which the maximum delay value is updated only in response to ECN marked packets. Other types of packet losses are interpreted as non-congestion losses. However, there may also be packet losses due to congestion situations when the ECN mark is missing because ECN marking is not possible in all cases. If there is a very large burst present in the network and the buffer is filled up, packets must be dropped instead of marked. CVIHIS can cope with this special case by combining all the above-described ways of dealing with non-congestion losses. If ECN is used and there are packet losses, the maxDelay value can be updated using equations 5-1 and 5-2.

5.3. Flow control as part of congestion control and CVIHIS

Flow control is a mechanism that allows the receiver to regulate the sending rate of the sender. Typically, every connection has a receive buffer on the receiving end. The main target of flow control is to prevent the overflow of this receive buffer. The buffer can be filled up if the packet processing power of the receiving machine is lower than the packet delivering power of the network, and lower than the sending power of the sender. This may happen if the receiving machine has a low speed processor, or if that processing power is used for other functions than emptying the receive buffer of the connection.

Because congestion control tries to prevent the buffer overflows of routers and flow control protects the receive buffer against overflows, these controls can be bound together in one unit (Garla and Kleinrock 1980). If it is supposed that congestion control is always implemented, there are three choices:

- flow control is not implemented at all;
- flow control is implemented as part of congestion control;
- flow control and congestion control are implemented separately.

If flow control is not implemented, it can affect the congestion state of the network. If reliable communication is used with packet retransmissions, the buffer overflow means that the packet must be discarded on the receiver side. This leads to retransmissions, which can increase the congestion risk inside the network.

If flow control is implemented as part of congestion control, only one cumulative feedback is delivered to the sender. Unfortunately, this kind of implementation has at least one significant disadvantage in relation to congestion control. If the sending rate is controlled by the state of the receive buffer for a long time, knowledge of the congestion state inside the network is lost on the sender side. After changing the sending rate from flow control limited transmission to congestion control limited transmission, the network can be underutilized for a long time before the additive increase policy reaches the sending rate allowed by congestion control. If congestion control and flow control are implemented separately, the sender is always aware of the state of the routers and the state of the receiving buffer. The sender gets two separate feedbacks and chooses the rate that is the minimum of the feedbacks obtained through the flow control and congestion control. However, changes from the flow control limited state to the congestion control limited state may increase the sending rate quickly without the presence of a moderate additive increase component.

It is possible to integrate flow control functions into CVIHIS. However, one could ask whether this integration is reasonable. For example, Kohler et al. (2006b) note that the DCCP protocol is not a flow control protocol. In addition, these kinds of applications typically drain the receive buffer at the sending rate used. If DASH is used, it typically tries to prevent the overflow of the application level receive buffer, and additional functions may only complicate the proper operation of DASH.

In principle, integrating flow control into CVIHIS can be done simply. The receive buffer can be divided into different parts in the same way as presented in Figure 4.4 for router queues. These different parts can be equipped with feedback responses, also shown in Figure 4.4. Thus, separate responses can be obtained for congestion control and flow control. At the receiving end, the integrated algorithm selects the response indicating the slower sending rate of these two responses and puts the corresponding feedback in the header of the acknowledgement message that is then sent to the sender. Unfortunately, this behavior can degrade the performance of the congestion control. If the sending rate is limited by the flow control, the sending rate can move far away from the rate determined by the congestion control. When control is switched back to congestion control, the network can be underutilized for a long time. However, CVIHIS continues to probe the queue levels of the network during the flow control period. The sender will learn the status of the network immediately after switching to the congestion control mode.

Changing from flow control to congestion control does not cause any problems to the backward-loading mode as there are no bandwidth demands in this mode. A CVIHIS-BLM connection can increase its sending rate without any hurry. Instead, the flow control mode and its slow sending rate can cause problems to the real-time mode of CVIHIS after switching back to the congestion controlled operation. Fortunately, there are some solutions that alleviate this problem. The minimum rate of CVIHIS still guarantees low-quality service. The receive buffer is filled to a high level because flow control has just been active. Therefore, it is not a major problem if the sending rate remains at a modest level for a while. This problem can also be alleviated by using a similar kind of bandwidth probe mechanism as proposed for the start phase of a connection. The probe mechanism can be activated after the mode change. One more flag bit would be needed in the header structure to detect this change. This bit advises whether the rate is controlled by flow control or congestion control.

5.4. Rate adaptation in two special cases

There are two special cases when the sending rate cannot be adapted in a normal way. The first case is known as a no-feedback situation. In this situation, the sender does not receive any ACK messages within a certain time. There can be many reasons for this. There may be problems with the client side application process. For example, the client application can be closed without an expedient control. There can also be connection breaks while the access points of a wireless connection are being changed. Above all, a bad congestion situation inside the network can generate this kind of situation. The congestion can occur either on a forward path or a backward path. The second special case is related to what is called the rate-limited period. In this case, the sending rate of the sender is significantly below the maximum allowed rate. This kind of situation typically appears when an application process cannot deliver enough data to the transport layer.

The no-feedback situation can be solved by decreasing the sending rate with a large step after the predefined silent period. CVIHIS can activate the multiplicative decrease step without using low-pass filtering after every such silent period. If the sending gap is multiplied by a factor of about 1.25-1.10, the rate decrease behavior is perhaps too modest for a bad congestion situation. However, if the decreasing step is taken repeatedly, the sending rate will decrease significantly after a couple of such silent periods. In addition, a factor of about 1.25-1.10 is more suitable for other types of short connection breaks.

It is a straightforward task to define the suitable length of a silent period with CVIHIS because the sender is aware of the round-trip properties of the connection. The sender has to be equipped with the same kind of smoothed round-trip time calculation algorithm as the TCP protocol. Based on this calculation, a no-feedback timer can be obtained using multiplication with a factor of about four. The argumentation behind this value is that it is twice as big as the factor of two used to calculate the RTO timer by the old version of the TCP RTO calculation algorithm. The rate-decreasing operation can wait a little longer than the packet retransmission operation. TFRC also uses a factor of 4 for the calculation of its no-feedback timer. Unfortunately, this no-feedback implementation would make the sender side implementation more complex.

The simplest solution regarding silent time definition is to use a certain constant value for this timer. There is, however, a drawback with this solution. A constant value does not take into consideration the different round-trip times of connections. A value like this should be about a few seconds so that it would be long enough for all connections. It would take a long time before connections would react to a bad congestion situation.

Tailored rate update rules must be applied during the rate-limited period because the network condition cannot be probed using the allowed sending rate. Instead, some lower sending rate is used. A good solution could be not to allow *the allowed sending rate* to be increased in the rate-limited state. In this state, the rate increase feedbacks are ignored while the rate decrease feedbacks are being handled and *the allowed sending rate* is decreased because there is at least an incipient congestion in the network. In such an incipient congestion state, the sending rate cannot be increased from the limited rate to the allowed rate by any large step when the rate-limited situation happens to end. Therefore, the decrease policy is used to make the increase step lower and lower.

5.5. Reverse path congestion control

Reverse path congestion control means that the amount of acknowledgement traffic is controlled. Typically, the number of acknowledgements is controlled by a parameter called an acknowledgement ratio. This ratio determines the number of received data packets after which an acknowledgement message is sent. There are also other kinds of control solutions. For example, TFRC sends an acknowledgement message after every round-trip time. There are many reasons why the reverse path may become congested. One reason is the asymmetry of network bandwidth. Especially, the access links of home connections are often this type of asymmetric links. There are also some bidirectional applications that generate a lot of traffic in both directions.

The need for reverse path congestion control is not an easy question. For example, the TCP protocol does not consider any kind of reverse path congestion control functions. The number of TCP ACKs can be reduced with the help of the delayed ACK mechanism but this mechanism is not controlled by the reverse path conditions. However, there has been quite a new proposal for TCP reverse path congestion control. Floyd et al. (2010) describe a possible congestion control mechanism for the ACK traffic of TCP.

The need for reverse path congestion control with CVIHIS is also a multifaceted question. CVIHIS already includes a basic solution to protect networks against reverse path congestion. The basic solution is the asymmetric nature of CVIHIS traffic. CVIHIS sends data only in one direction. Only small ACK packets without the application date are sent in the reverse direction. Therefore, the reverse path traffic is typically not more than 10% of the total data traffic. Without a considerable asymmetry, the CVIHIS ACK traffic cannot be the real cause of reverse path congestion. On the other hand, many acknowledgements may be dropped due to congestion on the reverse path. In such cases, CVIHIS may react too slowly to changes in network conditions because the sending rate of CVIHIS is updated by acknowledgements. Based on this fact, it might be desirable to equip CVIHIS with reverse path congestion control. If a no-feedback timer is used, this kind of solution for serious reverse path congestion situations is already in place because multiplicative decrease steps are automatically taken to reduce the sending rate. A reduction in the number of data packets also reduces the number of ACK messages. One more argument against CVIHIS reverse path congestion control is that it adds to the complexity of the CVIHIS sender.

If reverse path congestion control is used, the CVIHIS sender has to detect lost or ECN-marked ACK packets. The value of the ACK ratio parameter is derived on the basis of this detection. This value of R is sent to the receiver in a piggybacking manner using a CVIHIS data packet. After that, the CVIHIS receiver sends roughly one ACK packet for every R data packets. CVIHIS probes the queue levels of the routers along the connection path. The delayed cumulative ACK can be based on the latest data packet received from the network because it gives the latest information about the network state. There is, however, one exception when a delayed ACK cannot be used. If a packet loss is detected, a multiplicative decrease command should be sent immediately. If reverse path congestion control is used with CVIHIS, it is preferable to use small values for the ACK ratio. Values such as 1, 2, or 4 are suitable.

There are two reasons behind the recommendation of using a small ACK ratio. Firstly, as Van Jacobson explains in Jacobson et al. (1992), a problem like this is a signal-processing problem. The frequency of the observed signal is the rate at which the data packets are sent. If samples of this signal are taken at a slow rate, the signal may be sampled at too low a frequency. This violates Nyquist's criteria and may therefore cause errors. Another argumentation for using a low ACK ratio is that CVIHIS uses delay gradients for congestion control. The delay gradients lie open to traffic phase effects and random events. For example, at a certain moment, two other packets may be queued between the packets of a particular connection, while a little later there may be three or more other packets. Based on the delay gradients, one could draw the conclusion that delays would increase although the delay properties of the network may have remained the same and the traffic phase effect simply puts packets in the queue in a different way. If CVIHIS sends acknowledgements at a high frequency, CVIHIS's congestion control sometimes takes a step in the wrong direction. However, this is corrected by upcoming acknowledgements.

5.6. Problems of delay-based congestion control algorithms

Delay-based congestion control mechanisms have a few known problems that can affect their performance. The papers by La et al. (1999), Rodríguez-Pérez et al. (2011), and Fu et al. (2001) list and analyze these problems. Some solutions are also proposed. The common problems of delay-based congestion control mechanisms are:

- an inability to compete fairly against loss-based congestion control protocols;
- rerouting problems;
- persistent congestion;
- asymmetry problems;
- clock synchronization problems if one-way delay measurements are used.

In this section, the capability of CVIHIS to cope with these problems is examined, although the CVIHIS real-time mode is not a pure delay-based mechanism.

Competing against loss-based congestion control protocols is not a big problem for CVIHIS. In relation to the backward-loading mode of CVIHIS, it is in fact the main design principle that this mode gives bandwidth away to other connections. In contrast, the real-time mode of CVIHIS has to compete against loss-based congestion control protocols. In Chapter 4, when CVIHIS-RTM was simulated and tested against the loss-based TCP protocol, it was observed that the real-time mode of CVIHIS is in fact capable of competing against TCP. This mode can compete against loss-based algorithms because CVIHIS-RTM shifts its target delay area upwards when there is an incipient congestion in the network.

If the route of a connection is changed without an explicit signal from the network, the end host cannot detect it. If the new route has a shorter propagation delay, this does not cause any serious problem for CVIHIS because some packets will probably experience shorter one-way delay values and the minimum delay value will be updated. The maximum delay value will also be updated by the next packet drop. If the maximum value is calculated using an EWMA method, as presented in section 5.2, it will be updated gradually.

If the new route of the connection has a longer propagation delay, it can pose a problem for CVIHIS. The connection cannot know whether the increase in the delay is due to congestion in the network or a change of route. Without this knowledge, the end host will interpret the increased delay as a signal of congestion and the host will decrease its sending rate. Fortunately, the real-time mode of CVIHIS pushes the minimum delay value upwards continuously. The minimum delay value will be automatically updated based on the delay properties of the new route.

This kind of rerouting problem also means that it is perhaps useful to modify the backward-loading version of CVIHIS. Continuous shifting of the minimum delay value could also be used with the backward-loading mode. The shifting factor of this mode should be only slightly above the value of one because CVIHIS-BLM does not wish to compete against other connections. La et al. (1999) introduce another kind of solution for updating the minimum delay value. After N data packets are received, the receiver can check the smallest delay value among these packets. If the difference between the smallest delay value and the current minimum delay value is larger than a certain threshold for a certain number of consecutive times, the receiver interprets this as a change in the propagation delay. This change typically means that the route used through the network has changed. The threshold used in this comparison is connection-based. To obtain a suitable threshold, the current minimum delay value is multiplied by a certain predefined factor.

The rerouting properties of CVIHIS were tested using a simulation. This test was done using the real-time mode. The version used in this simulation did not use any of the improvements presented earlier in this section. The ability of CVIHIS-RTM to discover rerouting is based on pushing of the minimum delay value and updating of the maximum delay value. The maximum delay value is updated after every packet drop.

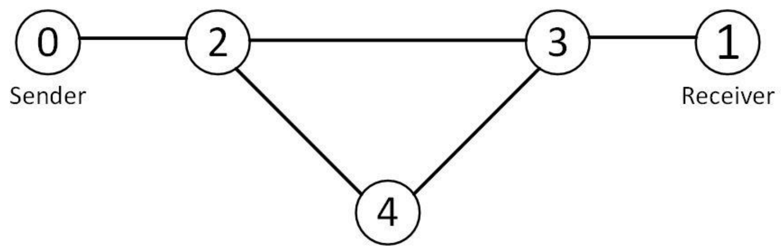


Figure 5.2 Network layout of the rerouting test

The network layout used in this simulation is presented in Figure 5.2. There are two possible routes between nodes 2 and 3. The default route was switched off twice during the simulation. Therefore, the traffic had to be switched to the backup route which traveled through node 4. The default route was switched off between seconds 80-150 and 220-320. The capacity of the default route was 700 kbps and the capacity of the backup route was 400 kbps. The simulation result of Figure 5.3 shows that CVIHIS-RTM can observe route changes and it can accommodate its sending rate according to the new route.

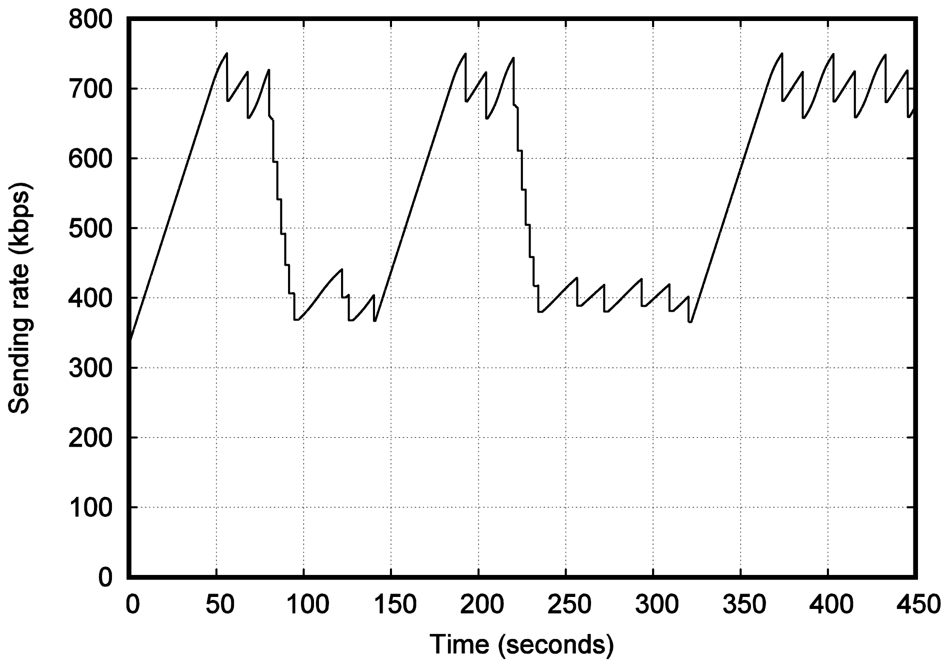


Figure 5.3 Rerouting test of CVIHIS-RTM

CVIHIS uses the minimum delay value to estimate the propagation delay of the route. If the connection overestimates the propagation delay due to an incorrect minimum delay value, persistent congestion can occur. For example, a new connection can enter the network when there is already an incipient congestion in the network. Due to the queuing delays caused by the other connections, the packets of the new connection may experience delays that are considerably larger than the actual propagation delay of the route. As a result, the minimum delay value of the new connection is set at a high level. The delay areas, presented in Figure 4.4, shift upwards for the new connection and the new connection tries to keep too many packets in the queue. The phenomenon can repeat after each new connection and may drive the network into a persistent congestion state.

La et al. (1999) present two proposals that alleviate the problem of persistent congestion. The first proposal is to employ RED in the routers. The second solution is the shifting of the minimum delay value. After a certain level of congestion, RED gateways start to drop incoming packets. Because packets are dropped independently, it is likely that more packets are dropped from the connections of higher sending rates. This forces those connections to reduce their sending rates. The reduction in sending rate is often made in a multiplicative decrease manner. Therefore, RED gateways alleviate the problem of persistent congestion. When the congestion level is low, connections can receive a fairly good estimate of the propagation delay without the help of RED.

When the network remains in the persistent congestion state, CVIHIS-RTM pushes its delay areas upwards by shifting the minimum delay value. This increases the congestion level of the network leading ultimately to packet drops. Many connections back off after these packet drops. They reduce their sending rates in a multiplicative decrease manner and the congestion level of the network falls. This allows CVIHIS to estimate the correct value of the minimum delay. Both of the solutions described alleviate the persistent congestion problem by generating packet losses. After the packet losses, normal multiplicative decrease behavior takes place. In fact, all events that generate packet losses are useful for solving this problem. The normal TCP congestion control behavior also generates packet losses. Therefore, the fact that CVIHIS shares the same resources with TCP connections is not necessarily negative.

The problems of asymmetry and clock synchronization are not discussed further because they were addressed in earlier parts of this thesis. The asymmetry problem was addressed when discussing reverse path congestion control. The clock synchronization problem was discussed in the previous chapter.

In summary, the shifting of the minimum delay value is a good choice when dealing with the typical problems of delay-based congestion control algorithms. It helps with all the above-mentioned problems with the exception of the asymmetry problem. The disadvantage of minimum delay value shifting is that the smoothness of the CVIHIS-RTM sending rate is not ideal.

5.7. Header structures of CVIHIS

Some header structures are needed for CVIHIS to deliver control information between the sender and the receiver. In this section, these header structures are examined with the help of a certain example. This example is the DCCP protocol, because different kinds of congestion control mechanisms can be used with DCCP. In theory, CVIHIS could be one of them although CVIHIS is designed in an open manner in this work without answering the question of which part of the protocol stack CVIHIS should reside in. If CVIHIS is integrated into some other protocol, similar header structures will be needed.

DCCP can offer appropriate header fields for the use of different kinds of congestion control mechanisms. The generic header structure of DCCP packets is presented in Figure 5.4. The header overhead of DCCP is minimized by using only a small generic header that contains the necessary information for all packet types. Any additional function needed by some particular packet type is implemented by adding a corresponding additional field after the generic header. After these fields, there is a possible list of options of variable length. The application data area follows the header if the packet type carries application data.

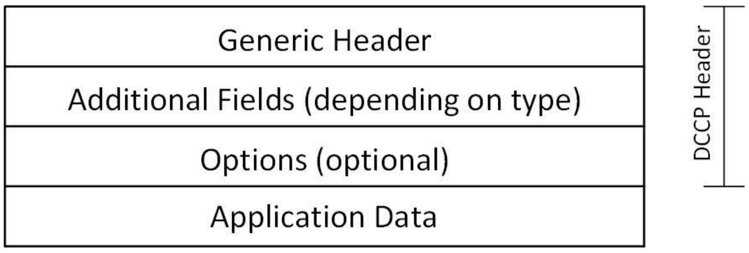


Figure 5.4 DCCP header structure

DCCP has ten packet types. In Figure 5.5, the specific structure of a forward path DCCP-DATA packet is presented. In the upper part of the figure, the generic header is shown. For example, the CCVal field is reserved for congestion control specific information. It can be used by the congestion control mechanism to encode some information into the packets. A possible option part comes after the generic header. For example, if the congestion control mechanism needs timestamps for delay or RTT measurements, a timestamp option can be added inside the option part of the DCCP-DATA packet. Since nowadays RTP is used for some real-time multimedia applications, Perkins (2010a) specifies a way to place RTP onto DCCP so that real-time applications can start to use the services provided by DCCP.

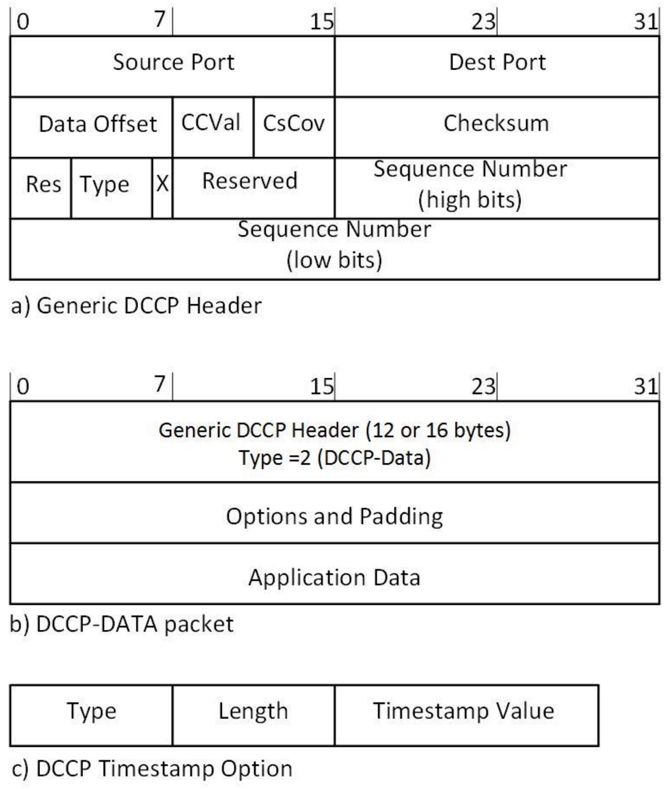


Figure 5.5 Header structure of the DCCP-DATA packet

In the case of CVIHIS, the required header functions and fields have to be recognized first. If the implementation is based on the basic mechanism introduced in Chapter 4, the data packet should have the following header fields:

- A field for the packet sequence number. Typically, 32 bits are reserved for this field.
- A timestamp field so that the packet sending time can be integrated into the packet. Typically, 32 or 64 bits are reserved for this field. The field of 32 bits is capable of expressing the clock resolution of 0.1 milliseconds used in the real network tests of this thesis.

The following header fields are needed for the acknowledgement packet:

- A field for the rate adaptation command. Four bits is enough for this field.
- A field enabling the calculation of the current round-trip time estimate. The solution is considered when the packet sending time is echoed back to the sender. Typically, 32 bits are reserved for this field.

All these fields are mandatory.

If all the candidate functions presented in this chapter are implemented, more header fields are needed. The following extra header fields are needed for the data packet:

- A field for the ACK ratio (optional). With the help of this field, the ACK ratio can be delivered to the receiver. Three bits is enough for this field.

- A flag bit for the special start algorithm (optional). With the help of this bit, the sender can tell which data packets are the probe packets of the start phase. The receive gap between these two packets has to be sent back to the sender.

The following extra header fields are needed for the acknowledgement packet:

- A sequence number field for the acknowledgements (mandatory). With the help of this field, the sender can recognize the lost ACK packets for the ACK ratio calculation. Typically, 32 bits are reserved for this field.
- A flag bit for the flow control (mandatory). This bit tells if the sending rate command of the particular ACK is based on flow control or congestion control.
- A field for the special start algorithm (optional). With the help of this field, the measured probe gap can be sent back to the sender. Typically, 32 bits are reserved for the field.

CIVHIS does not need many header fields for the basic mechanism. This is quite natural as one of the design targets was simplicity.

Because CVIHIS transfers data only in one direction, two types of DCCP packets are needed to control data transfer. The sender side sends DCCP-DATA packets and the receiver sends DCCP-ACK packets. The structure of DCCP-DATA packets was presented in Figure 5.5. The structure of DCCP-ACK packets is similar to the structure of DCCP-DATA packets except that there is one more field for the acknowledgment number. This field is implemented as an additional field that resides between the Generic DCCP header and the options.

Regarding the basic mechanism, the required header fields can be placed as explained below. In the DCCP-DATA packet, there is already a field for the packet sequence number. The timestamp field has to be put to the option part of the DCCP-DATA packet. The structure of the timestamp option was presented in Figure 5.5. Concerning the DCCP-ACK packet, the location of the rate adaptation command is somewhat problematic. In the generic DCCP header, there is the CCVal field for the congestion control specific information. This field is 4 bits long and, therefore, it is long enough for the rate adaptation command. Where header efficiency is concerned, the CCVal field is the right place for the rate adaptation command. However, Kohler et al. (2006b) state that this field can be used only by a data sender. If the work of Kohler et al. (2006b) is interpreted strictly, the rate adaptation command has to be put in the option part of the DCCP-ACK packet. In this case, a new option can be defined as DCCP allows the definition of congestion control specific options. The DCCP-ACK packet also includes a timestamp option for echoing the sending time of the data packet back to the sender.

Next, the extra header fields for the candidate functions have to be placed into the DCCP header structure. The ACK ratio field and special start algorithm flag can be put in the CCVal field of the DCCP-DATA packet. The CCVal field has four bits, which is sufficient for these functions. Please note that DCCP has a feature called the ACK ratio, which can be sent to the receiver using the feature negotiate function. DCCP endpoints can use Change and Confirm options to negotiate and agree on feature values. The feature negotiation usually happens at the connection initiation phase but it can also happen during the connection. However, feature negotiation options cannot be sent by the DCCP-DATA packets. Therefore, in the middle of the connection, feature negotiation has to be carried out using the DCCP-SYNC and DCCP-ACK packets. Obviously, this negotiation-based alternative would make the implementation more complex.

The candidate functions also need control information to be placed into the acknowledgement packet. There is already an acknowledgment number field in the DCCP-ACK packet for the required acknowledgement sequence number. A flag bit is required to report whether the sending rate command is based on flow control or congestion control. It is useful to combine this flag bit with the rate adaptation command and, therefore, the flag bit can be placed either in the CCVal field or the rate adaptation command option. The problem with the CCVal field is that five control bits would be needed whereas the CCVal field is only four bits long. Therefore, a solution must be chosen where a congestion control specific option is used to implement the rate adaptation command option. There is also a need for a timer type option. With the help of this option, the special start algorithm can deliver the value of the measured probe gap to the sender. This option can be formed in the same way as the timestamp option in Figure 5.5.

With the help of this example, we have also investigated how the DCCP protocol definition is suited for the use of a new congestion control mechanism. DCCP is able to implement all the required CVIHIS functions. If there is no other way to implement a certain function, it can be implemented by defining a congestion control specific option. Regarding the work presented in this thesis, there is at least one drawback related to the DCCP specification. The use of some header fields is defined in an overly restrictive way. The CCVal field is one example. Another example is that DCCP-DATA packets cannot be used for feature negotiation. Due to these restrictions, the implementation may become more complex. The header efficiency may also suffer. Based on the findings of this thesis, it is preferable to define these kinds of general-purpose protocols more openly. The actual code implementation of a certain mechanism can be responsible for the interpretation of the header fields.

5.8. Multicast properties of CVIHIS

The multicast properties of CVIHIS are discussed in this section because multicast is theoretically the right way to deliver real-time streaming in particular to a group of receivers. The TCP/DASH approach is not suitable for genuine multicast. Of course, the use of CDN networks for video distribution has somewhat reduced the need to implement genuine multicast. A multicast sending event forms a tree structure, such as the one shown in Figure 5.6. The root of the tree is the multicast server, which is the source of the traffic. Middle nodes reside under the root and form distribution paths with a different number of hierarchies. The receiver of the multicast data flow is a leaf on this tree structure. These leaf nodes can reside at different levels of the hierarchy.

The multicast tree structure can be interpreted in two ways. In the first case, the middle nodes are routers which form the multicast tree. In this case, all routers work in the multicast mode and packets are multicast *at the network layer*. If tunneling is used between two routers, there can also be non-multicast routers between these routers. In the second case, the middle nodes are distribution servers. In this case, there can be many routers between the two middle nodes. This multicast mode is also known as overlay multicast or end-system multicast. In the overlay multicast, an overlay network is built on top of available network services and packets are multicast *at the application layer* (Abad et al. 2004). Where the network layer is concerned, all routers now work in a normal unicast manner. The overlay multicast is quite common because IP multicast is still awaiting wide deployment. The overlay mode may be sometimes more desirable because multicast equipped with congestion control can cause high processing demands to routers (Cui et al. 2004).

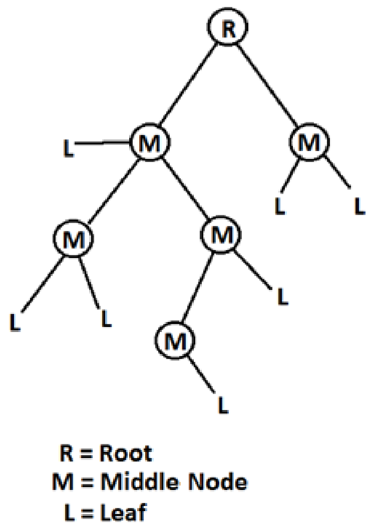


Figure 5.6 Multicast tree

As mentioned in Chapter 2, multicast congestion control has a filtering problem. CVIHIS has quite good properties against the filtering problem. There are only discrete numbers of congestion control feedbacks of CVIHIS and they are represented by numerical values. If congestion feedbacks are expressed in bytes or bit rates, filtering is much more challenging because the feedback space is continuous. Regarding the CVIHIS backward-loading mode, filtering is quite easy to implement. Because there are no tight bandwidth demands with this mode, the slowest feedback can be chosen from all the feedbacks. With the real-time mode, the problem is more complicated because there are bandwidth demands. There is no point using a slow sending rate if there are only a couple of receivers having problems with their connection paths. CVIHIS-RTM can use a simple mathematical formula to calculate the weighted feedback. In the weighted feedback, it is often reasonable to emphasize negative feedbacks more than positive feedbacks as it is wise to be more conservative than liberal.

There are two approaches to control congestion in multicast communication, i.e., single-rate and multi-rate approaches. In the case of single-rate multicast congestion control, the whole multicast tree uses the same sending rate at a certain moment. This single-rate multicast mode approach is especially suitable for the backward-loading mode of CVIHIS. In this way, content can be uploaded to cache servers. Because the leaf node of the multicast tree can probe the delay properties all the way from the sender to the leaf, it is sufficient that only the leaf nodes run the receiver side algorithm of CVIHIS. The middle nodes perform filtering based on the slowest feedbacks and deliver feedback information to the sender. Due to the tree structure of this kind of cache loading, only some leaf nodes can be chosen as representative nodes and only they have to send feedback information. There are already some proposals for this kind of single-rate multicast congestion control mechanism including delay and delay gradient based properties. Kumar and Singh (2012) and Singh et al. (2008) present two such approaches. With the help of delays, these two mechanisms can work in a proactive manner and thus reduce packet losses.

On the other hand, the multi-rate approach is more suitable for the real-time mode of CVIHIS. CVIHIS-RTM could use hierarchical data transmission or the DASH-like multi-version approach to implement the multi-rate approach. In hierarchical data transmission, a data stream is presented with the help of hierarchical layers. This hierarchical layer structure includes a base layer representing the transmitted data content in a basic quality, and several enhancement layers for the high quality representations. The multi-rate multicast transmission places high processing demands on the middle nodes. Therefore, we can suppose that this kind of data delivery is implemented using the overlay multicast mode where middle nodes are distribution servers instead of routers. In this kind of environment, CVIHIS can be implemented in a hop-by-hop manner. Any two nodes of the multicast tree can have their own CVIHIS session between them.

6 Conclusions and Future Work

To conclude the thesis, this chapter summarizes the main studies and results presented. Section 6.1 introduces some conclusions related to the impact of this work. Some deployment issues are discussed in section 6.2. A summary of the contributions of this thesis is given in section 6.3. This chapter also introduces some proposals for future work.

6.1. Conclusions related to the achievement of the objectives of this study

During the last decade, video type data services in their various forms have become very common. In certain parts of the network, this type of data transmission consumes over half of the network capacity. These applications usually work on top of the TCP protocol, or on top of the UDP protocol. TCP protocol has its weaknesses related to multimedia streaming. Using UDP means that no congestion control is used with this kind of communication. Several new congestion control and rate adaptation mechanisms have been developed for these services during recent years. There are mechanisms for low-priority services where the bandwidth is given away for the use of other connections after congestion situations. There are also mechanisms that compete with other connections for the use of the network bandwidth in a fair manner.

In this thesis, these two approaches are combined and an integrated mechanism has been developed. This mechanism includes two modes, the backward-loading mode and the real-time mode. The mechanism developed in this thesis is one of the first congestion control mechanisms integrating these two approaches into one implementation. First, the development and testing were presented in Chapter 4. Later, in Chapter 5, some elaboration work was presented to adjust the new mechanism to several kinds of special cases.

Several quite challenging targets were set for the new mechanism, described at the beginning of Chapter 4. An analysis is presented next of how these targets were met. The first target was that the new mechanism should include the two modes mentioned above. This target was reached as it was proved with the help of several tests that both modes worked at the fundamental level. The second target was that the algorithm should be receiver-based. This target was also fulfilled because there are very few conclusion procedures on the sender side compared to the receiver side.

It was also desired that the two modes of the new mechanism would be as convergent as possible. In relation to the basic implementation solutions, this target was reached extremely well. There are only minor differences between the code implementations of these modes. In fact, the algorithm of this thesis represents some kind of optimal solution where the level of convergence is concerned. By adding only a few code lines to the backward-loading mode of CVIHIS, a more aggressive implementation for the real-time mode was obtained. Furthermore, there was no need for code changes on the sender side. Due to these features, it is possible to switch seamlessly between the modes. For example, when there is enough data in the receive buffer, the real-time mode can be changed to the backward-loading mode. This kind of integrated solution works in a more sophisticated way than a solution in which the best low-priority congestion control algorithm and the best TCP-friendly algorithm are put to work together. Of course, if some elaborate functions are required, there will be more differences between the modes of CVIHIS.

As the next target, it was required that the real-time mode should be TCP-friendly. At the same time, however, it was desirable that the bandwidth of this mode would vary in a much smoother way than that of TCP. These targets were met, although, as is usual with this kind of solution, not perfectly. The developed mechanism could manage better regarding TCP friendliness in relation to short or long round-trip times. On the other hand, it was a deliberate decision to change the sending rate of CVIHIS based on the square root of the round-trip time instead of using TCP’s round-trip time approach. This met the target that the mechanism should be less biased against long round-trip times than TCP. There can also be problems with TCP friendliness if the capacity of the outgoing link is low compared to the sending rate of the CVIHIS connection. Indeed, it was known beforehand that TCP friendliness would be a challenge because CVIHIS and TCP differ from each other. CVIHIS uses an integrated delay-loss-based approach while TCP uses a pure loss-based approach. In addition, CVIHIS is a rate-based solution and TCP is a window-based solution.

The next target was that the mechanism facilitates DASH-based rate adaptation. Section 4.7 analyzes this target. The last target was that the mechanism should be stable and scalable. This target was also met. For example, the mechanism is scalable because it is an end-to-end based solution. There is no need for any kind of state information in routers. The receiver side implementation principle also supports the scalability target. The initial tests indicated that CVIHIS could manage even better if there were many connections on the network path. Where stability is concerned, it was found that the TCP-friendliness of CVIHIS was not overly sensitive to parameter settings. Small changes in the values of the adjusting parameters changed the simulation results only slightly.

6.2. Deployment considerations

One desirable feature of a new congestion control mechanism is that it is easy to deploy. This is also required from CVIHIS. The most important property regarding ease of deployment is that the new mechanism does not require any changes to routers. CVIHIS achieves this goal because there is no need for any CVIHIS specific operations in routers. However, if RED is used in routers, it is recommended to tailor RED parameters for CVIHIS.

It is also preferred that only one side of the endpoints requires changes so that there is no need for all endpoints to deploy a new mechanism. It is preferable that the server side be changed rather than the receivers because one server usually serves many receivers. Unfortunately, both endpoints have to be updated to be CVIHIS-compatible. However, this is not a big problem because updating is typically performed by the service platform of a certain video service. For example, some kind of set-top box or software module is given to the customer. Through this kind of service module, CVIHIS can also be protected against malicious users. When using this kind of service modules, there is no need to give the source code of CVIHIS to customers. It would be easy for malicious users to modify the source code of CVIHIS so that only rate increase commands are sent to the sender continuously.

There is another reason why it is not desirable that a new mechanism would perform actions in routers, which is the encryption of network traffic. As network security is an important issue nowadays, encryption is being adopted widely. In many cases, the whole IP payload is encrypted and intermediate nodes cannot read the protocol headers of upper layers. Therefore, it is possible that routers may not even know that the IP payload includes CVIHIS-type traffic. Any approach that needs updates to intermediate nodes destroys the end-to-end approach.

A new mechanism is also required to be flexible. Flexibility answers the question of whether a new mechanism can deal with different kinds of environments efficiently. These kinds of environments are, for instance, wireless networks with random packet losses and ADSL links with bandwidth asymmetry. For example, TCP Reno works well in wired networks but it makes erroneous interpretations in wireless networks, because random packet losses are interpreted as congestion packet losses (Lakshman et al. 2000). The flexibility of CVIHIS depends partly on how widely the enhancements of Chapter 5 are deployed. The basic version is not flexible in all possible cases but the enhancements proposed in Chapter 5 would make CVIHIS more suitable for heterogeneous environments. For example, the basic version can deal with bandwidth asymmetry but it needs enhancements to cope better with non-congestion packet drops.

6.3. Summary of research contribution

This thesis includes four main research contributions, which have been described in Chapters 2 to 5. Some main findings were made based on these contributions. The four contributions can be listed as follows:

1. Principles related to congestion control were introduced in Chapter 2. Only the issues relevant from the point of view of this thesis were discussed. Based on this contribution, the right choices could be made for the new mechanism.
2. Congestion control mechanisms and video streaming approaches used on the Internet were introduced in Chapter 3. The congestion control proposals important for this thesis were studied in more detail. These proposals include issues related to delay-based mechanisms, congestion control for multimedia applications, and TCP congestion control. Based on this work, it was found that the delay-based congestion control approach is suitable for low-priority services. Knowledge for developing a TCP-friendly congestion control mechanism was also obtained. It was found that TCP friendliness is a challenging issue.
3. A new tailored congestion control mechanism for video services was developed and tested, as described in Chapter 4. A lot of work was done using the NS-2 simulator in this part. A totally new protocol agent was developed and added to NS-2. The mechanism was also tested in our real-like test network. It was observed that it is possible to implement low-priority congestion control mechanisms using delay-based congestion control. It was also proven that, with moderate changes, this kind of low-priority algorithm could be made more aggressive. Due to this aggressiveness, the mechanism can also support TCP-friendly congestion control communication.
4. The basic mechanism was refined and elaborated for some special cases, as described in Chapter 4. It was found that the developed mechanism is quite resistant against the typical problems of delay-based congestion control mechanisms. The main reason for this resistance is that the minimum delay value of the real-time mode is not static. Instead, we try to push this minimum delay value upwards continuously. For example, rerouting and clock synchronization problems can be alleviated this way. It was also investigated how DCCP header structures adapt to the needs of the new congestion control mechanisms. It was found that some header definitions related to the DCCP protocol are perhaps too restrictive. When header structures of general-purpose protocols are defined, it is recommended that the definitions be made in a relaxed way to improve header efficiency.

6.4. Future work

There are some potential research directions for future work. The research work presented in this thesis has raised some questions that have to be answered. For example, in Chapter 5, some solutions for special cases were presented. More research is needed to clarify which of these solutions are useful enough and worth implementing. In this thesis, a fixed packet size was used in all simulation and test cases. There is a need to take into consideration heterogeneous packet sizes in future research. For example, rate adaptation could be based on the number of bytes rather than the number of packets.

There is also a need for research collaboration with specialists of other research areas. One such subject is how the rate adaptation behavior of the application is integrated with the mechanism of CVIHIS. It is an application-specific issue if the real-time mode is smooth enough for the needs of a particular application, because the sending rate of this mode oscillates slightly due to the pushing of the minimum delay value. Extensive utilization of DASH for video streaming brings its own features to this integration work. DASH facilitates this integration work but it also makes the environment even more heterogeneous. In addition, there are always some possibilities to fine-tune an algorithm like CVIHIS. A possible issue requiring fine-tuning is related to the result that CVIHIS drops somewhat more packets than expected based on its share of the bandwidth.

6.5. Final statements

The current state of the Internet sets limitations on the perfect operation of our congestion control mechanism. First of all, existing networks are heterogeneous. The round-trip times of connections vary greatly. There are several TCP versions that differ at least to some extent, with various DASH versions being used. Our study has shown that relatively small differences between the versions can pose major challenges for TCP friendliness. In addition, TCP’s congestion control strongly favors connections of short round-trip times, which also poses challenges for TCP friendliness.

Our understanding of Internet congestion control and the research results of this thesis provide the insight that congestion control can be implemented in an efficient way if there are only a few compatible congestion control mechanisms on the Internet. As long as we have this kind of heterogeneous environment, the only efficient congestion control solution seems to be the use of overprovisioned networks. This poses the question, however, whether overprovisioning is too simple a solution in the current technological world.

Bibliography

Abad, C., Yurcik, W. and Campbell, R. 2004. A Survey and Comparison of End-System Overlay Multicast Solutions Suitable for Network Centric Warfare. Proceedings of the SPIE, Vol. 5441, pp. 215-226.

Ahmad, S., Gohar, N. and Kamal, A. 2007. A Dynamic Congestion Control Mechanism for Real Time Streams over RTP. Advanced Communication Technology, The 9th International Conference, pp. 961-966.

Akan, Ö. 2004. On the throughput analysis of rate-based and window-based congestion control schemes. Computer Networks 44 (2004), pp. 701-711.

Akhshabi, S., Anantakrishnan, L., Dovrolis, C. and Begen, A. 2012. What happens when HTTP adaptive streaming players compete for bandwidth? In the Proceedings of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV’12), pp. 9-14.

Allman, M. 2003. TCP Congestion Control with Appropriate Byte Counting (ABC). IETF RFC 3465, 9 p.

Allman, M., Floyd, S. and Partridge, C. 2002. Increasing TCP's Initial Window. IETF RFC 3390, 15 p.

Allman, M., Paxson, V. and Blanton, E. 2009. TCP Congestion Control. IETF RFC 5681, 18 p.

Almes, G., Kalidindi, S. and Zekauskas, M. 1999. A One-way Delay Metric for IPPM. IETF RFC 2679, 19 p.

Appenzeller, G., Keslassy, I. and McKeown, N. 2004. Sizing Router Buffers. ACM SIGCOMM ’04, Portland, Oregon, September 2004. Also in Computer Communication Review, Vol. 34, No. 4, pp. 281-292.

Avrachenkov, K., Ayesta, U. and Piunovskiy, A. 2005. Optimal choice of the buffer size in the Internet routers. In the Proceedings of the 44th IEEE Conference on Decision and Control, Seville, Spain, pp. 1143-1148.

Bakre, A. and Badrinath, B. 1995. I-TCP: Indirect TCP for Mobile Hosts. Proceedings of the 15th International Conference on Distributed Computing Systems, May 1995, pp. 136-143.

Balakrishnan, H., Seshan, S. and Katz R. 1995. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. Wireless Networks, Vol. 1, Issue 4, pp. 469-481.

Begen, A., Akgul, T. and Baugher, M. 2011. Watching video over the web: Part 1: Streaming protocols. IEEE Internet Computing, Vol.15 Issue 2, pp. 54-63.

Bianzino, M., Chaudet, C., Rossi, D. and Rougier, J. 2012. A survey of green networking research. IEEE Commun. Surveys & Tutorials, Vol. 14 Issue 1, pp. 3-20.

Blumenthal, M. and Clark, D. 2001. Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World. ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 70-109.

Borri, M., Ferrarini, A. and Merani, M. L. 2007. RTP-Primal: A New Congestion Control Scheme Suitable for Smooth Multimedia Delivery. Wireless Communications and Networking Conference, WCNC 2007, IEEE, pp. 3752-3757.

Braden, R. 1989. Requirements for Internet Hosts -- Communication Layers. IETF RFC 1122, 116 p.

Braden, R., Zhang, L., Berson, S., Herzog, S. and Jamin, S. 1997. Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification. IETF RFC 2205, 112 p.

Braden, R., Clark, D., Crowcroft, J., David, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Petterson, L., Ramakrisman, K., Shenker, S., Wroclawski, J. and Zhang, L. 1998. Recommendations on Queue Management and Congestion Avoidance in the Internet. IETF RFC 2309, Informational, 17 p.

Brakmo, L., O'Malley, S. and Peterson, L. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. SIGCOMM '94 Proceedings of the Conference on Communications Architectures, Protocols and Applications, pp. 24-35.

Böttger, T., Cuadrado, F., Tyson, G., Castro, I. and Uhlig, S. 2016. Open Connect Everywhere: A Glimpse at the Internet Ecosystem through the Lens of the Netflix CDN. arXiv preprint arXiv:1606.05519, 14 p.

Cao, Z. and Zegura, E. 1995. Utility Max-Min: An Application-Oriented Bandwidth Allocation Scheme. INFOCOM '99 Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 2, pp. 793-801.

Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H. and Jacobson, V. 2016. BBR: Congestion-based congestion control. Queue - Network Congestion, Vol. 14 Issue 5, ACM, 34 pages.

Carlucci, G., De Cicco, L., Holmer, S. and Mascolo, S. 2016. Analysis and design of the google congestion control for web real-time communication (WebRTC). In the Proceedings of the 7th International Conference on Multimedia Systems, ACM, 12 pages.

Carofiglio, G., Muscariello, L., Rossi, D., Testa, C. and Valenti, S. 2013. Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol. Computer Networks, Vol. 57, Issue 8, pp. 1838-1852.

Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. and Wang, R. 2002. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. Wireless Networks 8/2002, pp. 467-479.

Cen, S., Pu, C. and Walpole, J. 1998. Flow and Congestion Control for Internet Media Streaming Applications. Proceedings of SPIE Multimedia Computing and Networking, pp. 250-264.

Chandra, E. and Subramani, B. 2010. A Survey on Congestion Control. Global Journal of Computer Science and Technology, Vol. 9, Issue 5, pp. 82-87.

Chiu, D. and Jain, R. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. Computer Networks and ISDN Systems 17/1989, pp. 1-14.

Choi, J., Reaz, A. and Mukherjee B. 2012 A Survey of User Behaviour in VoD Service and Bandwidth-Saving Multicast Streaming Schemes. IEEE Communications Surveys & Tutorials, Vol. 14, No. 1, pp. 156-169.

Choi, S. and Handley, M. 2009. Designing TCP-Friendly Window-based Congestion Control for Real-time Multimedia Applications. PFLDNeT 2009, The 7th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports, 6 pages.

Christiansen, M., Jeffay, K., Ott, D. and Smith, F. 2001. Tuning RED for Web Traffic. IEEE/ACM Transactions on Networking, Vol. 9, No. 3, pp. 249-264.

Cisco 2016a. Cisco Visual Networking Index: Forecast and Methodology, 2015-2020. Retrieved 2016-08-02, available from
http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

Cisco 2016b. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020. Retrieved 2016-08-02, available from
http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

Cui, Y., Li, B. and Nahrstedt, K. 2004. oStream: Asynchronous streaming multicast in application-layer overlay networks. IEEE Journal, Selected Areas in Communications, January 2004, pp. 91-106.

Damjanovic, D. and Welzl, M. 2011. An Extension of the TCP Steady-State Throughput Equation for Parallel Flows and Its Application in MulTFRC. IEEE/ACM Transactions on Networking, Vol. 19, No. 6, pp. 1676-1689.

De Cicco, L., Caldaralo, V., Palmisano, V. and Mascolo, S. 2013. Elastic: a client-side controller for dynamic adaptive streaming over http (dash). In Packet Video Workshop (PV), 2013 20th International, IEEE, pp. 1-8.

Denda, R., Banchs, A. and Effelsberg W. 2000. The Fairness Challenge in Computer Networks. In the Proceedings of the 1st International Workshop
on Quality of future Internet Services (QofIS 2000), pp. 208-220.

Dobrian et al. 2013. Understanding the impact of video quality on user engagement. Communications of the ACM, Vol. 56 Issue 3, pp. 91-99.

Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Herbert, T., Agarwal, Jain, A. and Sutin, N. 2010. An Argument for Increasing TCP's Initial Congestion Window. ACM SIGCOMM Computer Communications Review, Vol. 40, pp. 27-33.

Esteban, J., Benno, S. A., Beck, A., Guo, Y., Hilt, V. and Rimac, I. 2012. Interactions between HTTP adaptive streaming and TCP. In the Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 21-26.

Floyd S. 2000. Recommendation on using the “gentle” variant of RED. Retrieved 2013-09-24, available from http://www.aciri.org/floyd/red/gentle.html

Floyd, S., Arcia, A., Ros, D. and Iyengar, J. 2010. Adding Acknowledgement Congestion Control to TCP. IETF RFC 5690, 39 p.

Floyd, S. and Fall, K. 1999. Promoting the Use of End-to-End Congestion Control in the Internet. IEEE/ACM Transactions on Networking, Vol. 7, No. 4, pp. 458-472.

Floyd, S., Handley, M., Padhye, J. and Widmer, J. 2008. TCP Friendly Rate Control (TFRC): Protocol Specification. IETF RFC 5348, 58 pages.

Floyd, S. and Jacobson, V. 1991. Traffic phase effect in packet-switched gateways. SIGCOMM Computer Communication Review 21(2), pp 26-42.

Floyd, S. and Jacobson, V. 1993. Random early detection gateways for congestion avoidance. IEEE/ACM Transaction on Networking 1(4), pp. 399-413.

Floyd, S. and Kohler, E. 2003. Internet Research Needs Better Models.
ACM SIGCOMM Computer Communication Review, Vol. 33, Issue 1, pp. 29-34.

Floyd, S. and Kohler, E. 2007. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. IETF RFC 4828, 46 p.

Fu, C. P. 2001. TCP Veno: End-to-end Congestion Control Over Heterogeneous Networks. A thesis for the degree of doctor of philosophy, The Chinese University of Hong Kong, 119 p.

Fu, C. P., Chung, L. C. and Liew, S.C. 2001. Performance Degradation of TCP Vegas in Asymmetric Networks and its Remedies. Proceedings of the IEEE International Conference on Communications, pp. 3229-3236.

Fu, C. P. and Liew, S. C. 2003. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Network. IEEE Journal on Selected Areas in Communications, Vol. 21, No. 2, pp. 216-228.

Garla, M. and Kleinrock, L. 1980. Flow Control: A Comparative Survey. IEEE Transactions on Communications, April 1980, pp. 553-574.

Garret, M. and Willinger, W. 1993. Analysis, Modeling and Generation of Self-Similar VBR Video Traffic. ACM SIGCOMM '94, Proceedings of the Conference on Communications Architectures, Protocols and Applications, pp. 269-280.
Gupta, M. and Singh, S. 2003 Greening of the Internet. ACM SIGCOMM '03, Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 19-26.

HA, S., Rhee, I. and Xu, l. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel archive, Vol. 42, issue 5, pp. 64-74.

Hassan, M. and Jain, R. 2004. High Performance TCP/IP Networking. Prentice Hall, 383 pages.

Hayes, D. and Armitage, G. 2011. Revisiting TCP Congestion Control using Delay Gradient. NETWORKING'11, Proceedings of the 10th International IFIP TC 6 Conference on Networking, Vol. 2, pp. 328-341.

Henderson, D., Floyd, S., Gurtov, A. and Nishida, Y. 2012. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 6582, 16 p.

Hoßfeld, T., Schatz, R., Biersack, E. and Plissonneau, L. 2013. Internet video delivery in Youtube: From traffic measurements to quality of experience. In Data Traffic Monitoring and Analysis, pp. 264-301.

Hoe, J. 1996. Improving the Start-up Behaviour of a Congestion Control Schema for TCP. ACM SIGCOMM '96, Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 270-280.

Hsu, W. and Lo, C. 2014. QoS/QoE mapping and adjustment model in the cloud-based multimedia infrastructure. IEEE Systems Journal, Vol. 8 Issue 1, pp. 247-255.

Hu, W., Wang, Z. and Sun, L. 2016. A Measurement Study of TCP Performance for Chunk Delivery in DASH. arXiv preprint arXiv:1607.01172, 11 p.

Huang, Y. and Guerin, R. 2005. Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger? ICNP 2005, Proceedings of the 13th IEEE International Conference on Network Protocols, pp. 225-235.

Huang, T. Y., Handigol, N., Heller, B., McKeown, N. and Johari, R. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In the Proceedings of the 2012 ACM Conference on Internet Measurement, pp. 225-238.

Jacobson, V. 1988. Congestion avoidance and control. ACM SIGCOMM '88, Symposium Proceedings on Communications Architectures and Protocols, pp. 314-329.
Jacobson, V., Braden, R. and Borman, D. 1992. TCP Extensions for High Performance. IETF RFC 1323, 37 p.

Jain, R., Chiu, D. M. and Hawe, W. R. 1984. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Hudson, MA: Eastern Research Laboratory, Digital Equipment Corporation, 38 p.

Jain, R. 1989. A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. ACM SIGCOMM, Computer Communications Review, October 1989, Vol. 19, Issue 5, pp. 56-71.

Jain, R. 1990. Congestion Control in Computer Networks: Issues and Trends. IEEE Network Magazine, May 1990, Vol. 4, Issue 3, pp. 24-30.

Jansen, S. and McGregor, A. 2006. Performance, validation and testing with the Network Simulation Cradle. MASCOTS 2006, 14th IEEE International Symposium on Modeling, Analysis, and Simulation, pp. 355-362.

Jiang, J., Sekar, V. and Zhang, H. 2012. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, pp. 97-108.

Juluri, P., Tamarapalli, V. and Medhi, D. 2016. Measurement of quality of experience of video-on-demand services: A survey, IEEE Communications Surveys & Tutorials, Vol. 18 Issue 1, pp. 401-418.

Karn, P. and Partridge, C. 1987. Improving Round-Trip Time Estimates in Reliable Transport Protocols. ACM SIGCOMM '87, Vol. 17, No. 5, pp. 66-74.

Katabi, D., Handley, M. and Rohrs, C. 2002. Congestion Control for High Bandwidth-Delay Product Networks. ACM SIGCOMM Computer Communication Review, Proceedings of the 2002 SIGCOMM conference, Vol. 32, Issue 4, pp. 89-102.

Keshav, S. 1991. A Control-Theoretic Approach to Flow Control. ACM SIGCOMM '91, Proceedings of the conference on Communications architecture & protocols, pp. 3-15.

Kim, K., Jeongb, K., Kanga, C. and Seokc, S. 2010. A transmission control SCTP for real-time multimedia streaming. Computer Networks Vol. 54, Issue 9, pp. 1418-1425.

Kim, H. J., Yun, D. G., Kim, H. S., Cho, K. S. and Choi, S. G. 2012. QoE assessment model for video streaming service using QoS parameters in wired-wireless network. In Advanced Communication Technology (ICACT), IEEE, pp. 459-464.

Kohler, E. and Floyd, S. 2006. Profile for Datagram Congestion Control Protocol (DCCP), Congestion Control ID 2: TCP-like Congestion Control. IETF RFC 4341, 20 pages.

Kohler, E., Handley, M. and Floyd, S. 2006a. Designing DCCP: Congestion Control Without Reliability. ACM SIGCOMM '06, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 27-38.

Kohler, E., Handley, M. and Floyd, S. 2006b. Datagram Congestion Control Protocol (DCCP). IETF RFC 4340, 129 p.

Kua, J., Armitage, G. and Branch, P. 2017. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP. IEEE Communications Surveys & Tutorials, 25 p.

Kumar, S. and Singh, K. 2012. Efficient Single Rate based Multicast Congestion Control. IJCA Proceedings on National Conference on Innovative Paradigms in Engineering and Technology (NCIPET 2012), pp. 1-4.

Kurose, J. and Ross, K. 2017. Computer Networking: A Top-Down Approach. 7th Edition, Pearson International Edition, 852 p.

Kuzmanovic, A. and Knightly, E, 2006. TCP-LP: low-priority service via end-point congestion control. IEEE/ACM Transactions on Networking, Vol. 14, Issue 4, pp. 739-752.

La, R., Walrand, J. and Anantharam, V. 1999. Issues in TCP Vegas. Retrieved 2013-05-10, available from
http://www.eecs.berkeley.edu/~ananth/1999-2001/Richard/IssuesInTCPVegas.pdf, 17 p.

Lai, K. and Baker, M. 2000. Measuring link bandwidths using a deterministic model of packet delay. ACM SIGCOMM '00, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 283-294.

Lakshman, T., Madhow, U. and Suter, B. 2000. TCP/IP Performance with Random Loss and Bidirectional Congestion. IEEE/ACM Transactions on Networking, Vol. 8, No. 5, pp. 541-555.

Li, Q. and Chen, D. 2005. Analysis and Improvement of TFRC Congestion Control Mechanism. Wireless Communications, Networking and Mobile Computing, 2005, International Conference, Vol. 2, pp. 1149-1153.

Liang, S. and Cheriton, D. 2002. TCP-RTM: Using TCP for Real Time Applications. ICNP 2002, IEEE International Conference of Network Protocols, 20 p.

Lundin, H., Holmer, S. and Alvestrand, H. 2012. A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web. IETF Informational Draft, Version 3, October 2012, 14 pages. Retrieved 2013-09-20, available from https://tools.ietf.org/html/draft-alvestrand-rtcweb-congestion-00

Ma, K. J., Bartoš, R. and Bhatia, S. 2011. A survey of schemes for Internet-based video delivery. Journal of Network and Computer Applications, Vol. 34 Issue 5, pp. 1572-1586.

Man, C., Hasegawa, G. and Murata, M. 2006. ImTCP: TCP with an Inline Measurement Mechanism for Available Bandwidth. Journal Computer Communications archive, Vol. 29, Issue 10, pp. 1614-1626.

Manfredi, S., Oliviero, F. and Romano, S. 2013. A distributed control law for load balancing in content delivery networks. IEEE/ACM Transactions on Networking (TON), Vol. 21 Issue 1, pp. 55-68.

Mansy, A., Ver Steeg, B. and Ammar, M. 2013. Sabre: A client based technique for mitigating the buffer bloat effect of adaptive video flows. In the Proceedings of the 4th ACM Multimedia Systems Conference, pp. 214-225.

Mathis, M. and Cheng, Y. 2013. Proportional Rate Reduction for TCP. IETF RFC 6937, 16 p.

Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A. 1996. TCP Selective Acknowledgment Options. IETF RFC 2018, 12 p.

Matrawy, A. and Lambadaris, I. 2003. A Survey of Congestion Control Schemes for Multicast Video Applications. IEEE Communications Surveys & Tutorials, Vol. 5, Issue 2, pp. 22-31.

Meddeb, A. 2010. Internet QoS: Pieces of the puzzle. IEEE Communications Magazine, Vol 48, Issue 1, pp. 86-94.

Menth, M., Briscoe, B. and Tsou, T. 2012. Precongestion notification: new QoS support for differentiated services IP networks. IEEE Communications Magazine, Vol. 50, Issue 3, pp. 94-103.

Mondal et al. 2017. Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption. The 27th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video, 6 pages.

Nagle, J. 1984. Congestion control in IP/TCP internetworks. IETF RFC 896.

Nichols, K. and Jacobson, V. 2012. Controlling queue delay. Communications of the ACM, Vol. 55, Issue 7, pp. 42-50.

NS2 2013. A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, NS-2 Manual. Retrieved 2013-03-24, available from http://www.isi.edu/nsnam/ns/doc/index.html

Padhye, J., Firoiu, V., Towsley, D. and Kurose, J. 1998. Modelling TCP throughput: A simple model and its empirical validation. ACM SIGCOMM ’98, Conference on Applications, technologies, architectures, and protocols for computer communication, pp. 303-314.

Paudyal, P., Battisti, F. and Carli, M. 2014. A study on the effects of quality of service parameters on perceived video quality. In Visual Information Processing (EUVIP) 2014, IEEE, pp. 1-6.

Paxson, V., Allman, M., Chu, J. and Sargent, M. 2011. Computing TCP's Retransmission Timer. IETF RFC 6298, 11 p.

Perkins, C. 2010a. RTP and the Datagram Congestion Control Protocol (DCCP). IETF RFC 5762, 16 p.

Qazi, I. and Znati, T. 2011. On the design of load factor based congestion control protocols for next-generation networks. ELSEVIER Computer Networks, Vol. 55, Issue 1, pp. 45-60.

Ramakrishnan, K., Floyd, S. and Black, D. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. IFTF RFC 3168, 63 p.

Rejaie, R., Handley, M. and Estrin, D. 1999. RAP: An End-to-end Rate-based Congestion Control Mechanism for Real time Streams in the Internet. INFOCOM '99, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, pp. 1337-1345.

Rhee, I., Ozdemir, V. and Yu, Y. 2000. TEAR: TCP emulation at receivers – flow control for multimedia streaming. Department of Computer Science of North Carolina State University, retrieved 2013-02-12, available from http://lanada.kaist.ac.kr/pubs/ tear.pdf

Rhee, I. and Xu, L. 2007. Limitations of Equation-based Congestion Control. IEEE/ACM Transactions on Computer Networking, Vol. 15, Issue 4, pp. 852–865.

Rikli, N. 2011. Self-similarity and stationarity of increments in VBR video. King Saud University Journal of King Saud University - Computer and Information Sciences 4/2012, pp. 7-16, retrieved 2013-01-20, available from www.ksu.ede.sa

Rodríguez-Pérez, M., Herrería-Alonso, S., Fernández-Veiga, M. and López-García, C. 2011. Common problems in delay-based congestion control algorithms: a gallery of solutions. European Transactions on Telecommunications, Vol. 22, Issue 4, pp. 168-178.

Ros, D. and Welzl, M. 2013. Less-than-best-effort service: A survey of end-to-end approaches. IEEE Communications Surveys & Tutorials, Vol. 15, No. 2, pp. 898-908.

Sahinoglu, Z. and Tekinay, S. 1999. On Multimedia Networks: Self-Similar Traffic and Network Performance. IEEE Communications Magazine, January 1999, PP. 48-52.

Saltzer, J., Reed, D. and Clark, D. 1984. End-to-end arguments in system design. ACM Transactions on Computer Systems, Vol. 2, Issue 4, pp. 277-288.

Savage, S., Cardwell, N., Wetheral, D. and Anderson, T. 1999. TCP Congestion Control with a Misbehaving Receiver. ACM SIGCOMM, Computer Communication Review, Vol. 29, Issue 5, pp. 71-78.

Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V. 2003. RTP: A Transport protocol for Real-Time Applications. IETF RFC 3550, 104 p.

Seufert, M., Egger, S., Slanina, M., Zinner, T., Hobfeld, T. and Tran-Gia, P. 2015. A survey on quality of experience of HTTP adaptive streaming. IEEE Communications Surveys & Tutorials, Vol. 17 Issue 1, pp. 469-492.

Shalunov, S., Hazel, G., Iyengar, J. and Kuehlewind, M. 2012. Low Extra Delay Background Transport (LEDBAT). IETF RFC 6817, 25 p.

Shenker, S. 1995. Fundamental Design Issues for the Future Internet. IEEE Journal on Selected Areas in Communications, Vol. 13, Issue 7, pp. 1176-1188.

Shin, M., Park, M., Oh, D., Kim, B. and Lee, J. 2011. Clock Synchronization for One-way Delay Measurement: A Survey. In the Proceedings of the 3rd International Conference on Advanced Communication and Networking. pp. 1-10.

Singh, K., Yadav, R., Manjul, M. and Dhir, R. 2008. Bandwidth Delay Quality Parameter Based Multicast Congestion Control. IEEE ADCOM 2008, the 16th International Conference on Advanced Computing and Communications, pp. 399-405.

Sisalem, D. and Wolisz, A. 2000. LDA+: A TCP-Friendly Adaptation Scheme for Multimedia Communication. ICME 2000, IEEE International Conference on Multimedia and Expo, pp. 1619-1622.

Sivaraman, A., Winstein, K., Thaker, P. and Balakrishnan, H. 2104. An Experimental Study of the Learnability of Congestion Control. In the Proceedings of the 2014 ACM Conference on SIGCOMM, pp. 479-490.

Sodagar, I. 2011. The mpeg-dash standard for multimedia streaming over the internet. IEEE MultiMedia, Vol. 18 Issue 4, pp. 62-67.

Spring, N., Wetherall, D. and Ely, D. 2003. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. IETF RFC 3510, 13 pages.

Srivastava, V., Neel, J., MacKenzie, A., Menon, R., DaSilva, L., Hicks, J., Reed, J. and Gilles, R. 2005. Using Game Theory to Analyze Wireless Ad Hoc Networks. IEEE Communications Surveys & Tutorials, Vol. 7, Issue 4, pp. 46-56.

Staelens, N., Pinson, M. H., Corriveau, P., De Turck, F. and Demeester, P. 2015. Measuring video quality in the network: from quality of service to user experience. In the 9th International Workshop on Video Processing and Consumer Electronics (VPQM 2015), pp. 5-6.

Stallings, W. 2013. Data and Computer Communications. 10th Edition, Prentice Hall, 912 p.

Talaat, M., Attiya, G. and Kou, M. 2013. Enhanced TCP-friendly rate control for supporting video traffic over internet. Canadian Journal of Electrical and Computer Engineering, Vol. 36, Issue 3, pp. 135-140.

Tanenbaum, A. and Wetherall, D. 2010. Computer Networks. 5th Edition, Prentice Hall, 960 p.

tc 2016. tc(8) - Linux manual page. Retrieved 06.06.2016 from http://man7.org/linux/man-pages/man8/tc.8.html

Tian, Y., Xu, K. and Ansaru, N. 2005. TCP in Wireless Environments: Problems and Solutions. Communications Magazine, IEEE, Vol. 43, Issue 3, pp. 27-32.

Touch, J. 2012. Automating the Initial Window in TCP. IETF Internet Draft, 10 pages. available from http://www.isi.edu/touch/pubs/draft-touch-tcpm-automatic-iw-03.txt

Tsao, S., Lai, Y. and Lin Y. 2007. Taxonomy and Evaluation of TCP-Friendly Congestion-Control Schemes on Fairness, Aggressiveness, and Responsiveness. IEEE Network, IEEE, Vol. 21, Issue 6, pp. 6-15.

Vihervaara, J. and Loula, P. 2014. Delay-based Congestion Control Mechanism for Video Services, Mechanism including Backward Loading and Real-time Modes. WEBIST 2014, In Proceedings of the 10th International Conference on Web Information Systems and Technologies, Vol.1, pp. 127-134.

Vihervaara, J. and Loula, P. 2015. Dual-Mode Congestion Control Mechanism for Video Services. ICIMT 2015, The 7th International Conference on Information and Multimedia Technology.

Vihervaara, J., Loula, P. and Tuominen, T. 2016a. Performance Gains from Web Performance Optimization - Case Including the Optimization of Webpage Resources in a Comprehensive Way. WEBIST 2016, In Proceedings of the 12th International Conference on Web Information Systems and Technologies, Vol.1, pp. 188-193.

Vihervaara, J., Loula, P. and Alapaholuoma, T. 2016b. Dual-Priority Congestion Control Mechanism for Video Services - Real Network Tests of CVIHIS. KMIS 2017, In Proceedings of the 8th International Joint Conference on Knowledge Management and Information Sharing, Vol 3, pp. 51-59.

Vojnovic, M., Boudec, J. and Boutremans, C. 2000. Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times. INFOCOM 2000. 19th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, pp. 1303-1312.

Wang, H. and Williamson, C. 1998. A New Scheme for TCP Congestion Control: Smooth-Start and Dynamic Recovery. MASCOT ’98, Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 69-76.

Wang, B., Kurose, J., Shenoy, P. and Towsley, D. 2008a. Multimedia streaming via TCP: An analytic performance study. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Vol. 4 Issue 2, 22 p.

Wang, R., Gutha, B. and Rapet, P. 2008b. Window-based and rate-based transmission control mechanisms over space-Internet links. IEEE Transactions on Aerospace and Electronic Systems, Vol. 44, No. 1, pp. 157-170.

Wang, Y., Even, R., Kristensen, T. and Jesup, R. 2011. RTP Payload Format for H.264 Video. IETF RFC 6184, 101 p.

Wang, Z. and Crowcroft, J. 1992. Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm. ACM SIGCOMM, Computer Communication Review, Vol. 22, Issue 2, pp. 9-16.

Wei, D., Jin, C. and Low, S. 2006. FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Transactions on Networking, Vol. 14, Issue 1, pp. 1246-1259.

Welzl, M. 2005. Network Congestion Control, Managing Internet Traffic. Wiley Series in Communications Networking & Distributed Systems, 263 p.

Winstein, K. and Balakrishnan, H. 2013 TCP ex Machina: Computer-Generated Congestion Control. ACM SIGCOMM Computer Communication Review, Vol. 43, Issue 4, pp. 123-134.

Widmer, H., Denda, R. and Mauve, M. 2001. A Survey on TCP-Friendly Congestion Control. IEEE Network, Vol. 15, Issue 3, pp. 28-37.

XGRAPH General Purpose 2-D Plotter 2013. Retrieved 2013-03-25, available from http://www.xgraph.org/#anch3b

Xiao, X. and Ni, L. 1999. Internet QoS: A Big Picture. IEEE Network, Vol. 13, Issue 2, pp. 8-18.

Yang, Y., Kim, M. and Lam, S. 2001. Transient Behaviours of TCP-friendly Congestion Control Protocols. INFOCOM 2001, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 3, pp. 1716-1725.

Yang, Y. and Lam, S. 2006. Internet Multicast Congestion Control: A Survey. In Proceedings of the International Conference on Telecommunications (ICT’00), Acapulco, Mexico.

Zhang, Y., Ansari, N., Wu, M, and Yu, H. 2012. AFStart: An Adaptive Fast TCP Slow Start for Wide Area Networks. IEEE ICC 2012, Communication QoS, Reliability and Modeling Symposium, pp. 1260-1264.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-4124-7
ISSN 1459-2045