Equation-Based Layered Multicast Congestion Control

Gian Donato Colussi

Helsinki November 2nd 2004

MS Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

C-2004-71

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Gian Donato Colussi

Työn nimi — Arbetets titel — Title
Equation-Based Layered Multicast Congestion Control

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MS Thesis	November 2nd 2004	102 pages

Tiivistelmä — Referat — Abstract

We investigate the applicability of an equation-based approach to estimating TCP-compatible data rate for layered multicast sessions with very slowly varying sending rates and develop mechanisms to improve layer subscription decisions at the receiver. A layered multicast session consists of a sender emitting data to a number of cumulative layers and a set of receivers subscribed to a subset of the layers. Receivers estimate the loss fraction, the round-trip time, and use a TCP-model to estimate a TCP-compatible data rate. A receiver uses the computed rate for layer join and leave decisions. We conducted a series of testbed and simulation experiments under a variety of conditions to investigate the TCP-Friendly Rate Control (TFRC) behavior in a scenario with constant bit rate sender. We show that the computed loss event rate and the calculated rate depend on the sending rate compared to the normal equation following TFRC. We modified the loss estimation algorithm and show that using true packet loss as congestion measure cancels the dependency and gives accurate estimates of congestion. We also discuss possible subscription level management strategies using the sending rate independent estimate of TCP-compatible data rate. We developed a multicast congestion control protocol and implemented one of the discussed strategies. Using simulation, we show that in many scenarios it efficiently avoids oscillations and subscribes the fair layer.

ACM Computing Classification System (CCS):
C.2.2 [Network Protocols]
C.2.5 [Local and Wide-Area Networks]
C.4 [Performance of Systems]

Avainsanat — Nyckelord — Keywords
Multimedia Networking, TCP, Multicast, Congestion Control

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpulan tiedekirjasto, sarjanumero C-2004-71

Muita tietoja — övriga uppgifter — Additional information

Acknowledgments

This MS thesis was conducted at the Multimedia Communications Lab (KOM)[1] at the University of Darmstadt in Germany under Erasmus student exchange programme.

I wish to express my gratitude to professor Ralf Steinmetz for providing me the possibility to work in a motivating and challenging environment at the KOM lab. I am also grateful to professor Kimmo Raatikainen at the Department of Computer Science at the University of Helsinki for his patience at times when my piece of work was progressing slowly.

Dipl.-Ing. Ivica Rimac, my advisor and mentor at KOM, encouraged me to tackle challenges which, at first, seemed insurmountable. His analytical approach illuminated points I had been wrestling with, diminishing them to tractable size. I learned much from him and hope to have captured some of his analytical way of thinking.

During my stay in Darmstadt I got into an accident, in which my knee was injured. Consequently, I needed an operation at hospital plus intensive rehabilitation. Now, physically disabled, I was worried how to get on with everyday routines, not least how to complete my thesis. At this point, I was fortunate to be strongly supported and encouraged by Brigitte Astheimer, the university student exchange coordinator. I can not overestimate her help.

In search of good medical treatment I was excessively supported by Markus Ochs, who turned out to be truly a friend in need. Whatever needed to be done, Markus was there to help. As a scout, he was prepared.

Mz & Fz, kiitos lehtileikkeistä ym.

[1] http://www.kom.tu-darmstadt.de/

Contents

1 Introduction

The dominant protocol in the Internet is TCP. The success of TCP in rapid growth of traffic, applications, and topology is much due to its robust congestion control. Network congestion occurs in routers when the incoming data arrival rate grows beyond the output link capacity. Uncontrolled traffic in a congested network may lead to a state called *congestion collapse* where practically no useful data gets through the network. To keep the network operational in presence of congested links, sources must lower the sending rate to the level of the network capacity, or below it.

Current TCP congestion control has its roots in late 80's [Jac88]. TCP was originally designed for reliable bulk data transport that can't tolerate data loss but can tolerate some delay and variation of delay. However, TCP congestion control is suboptimal for purposes such as streaming video and audio, that can not tolerate large transmission rate variations, or multicast where acknowledgments from all receivers is not scalable (for large groups the traffic would be dominated by the acknowledgments). Smoother control is required for streaming media flows and mechanisms that do not require explicit feedback from all recipients are required for multicasting.

Given the overwhelming popularity of TCP it is very likely that any new congestion control protocol operating in the public best-effort Internet will have to coexist with TCP. Birth of TCP-compatible paradigm [BB98] has spawn efforts to develop other congestion control protocols that could peacefully coexist with TCP and provide more suitable control for application domains that can't benefit from TCP congestion control. Among these efforts are TCP-Friendly Rate Control (TFRC) [FHPW00b] and its derivatives to multicast [WH01, KB03], and Datagram Congestion Control Protocol framework [KHF04]. TFRC is based on an analytical model of TCP long-term throughput and its objective is to provide smooth TCP-compatible congestion control for unicast multimedia streams. DCCP is an attempt to develop a standard framework for unicast multimedia flows. DCCP is

not limited to any single congestion control protocol, rather it may be extended with different congestion control protocols suitable for slightly different application domains, e.g., Voice over IP or TV over IP.

1.1 Motivation

Multicasting is an efficient technology to reach a large audience for real-time streaming media where the same content is delivered to many recipients; there may be thousands of receivers in a multicast session. In a layered receiver-driven congestion control approach the session is split into cumulative data layers to which the sender (we restrict our discussion to single-source sessions) emits the data independent of the network congestion state. The cumulative organization means a predefined order of subscription all receivers follow. It also means that the data on a layer i is addition to data on layer j for all $i > j$. The congestion control is performed by each receiver subscribing[2] a subset of the offered layers. Receiver-driven layered multicast congestion control actually reduces to two decisions: (1) whether to subscribe the next layer, and (2) whether to unsubscribe the current maximum layer.

The fundamental difference between unicast and multicast congestion control is the closed-loop and open-loop nature of the control, respectively. In unicast communication, two end-systems constantly communicate with each other, one sending data and, the other providing feedback, and thus forming a closed-loop control. In multicast, communication end-systems form an open-loop control where a receiver only rarely sends feedback to the sender, if at all. A closed-loop congestion control architecture is not necessarily applicable to an open-loop scenario. TFRC derivatives to multicast implicitly assume that the closed-loop algorithm is applicable to an open-loop scenario. To say it differently: they assume that TFRC algorithm estimates a good TCP-compatible rate when the actual data rate and

[2]We use terms subscribe and unsubscribe to mean join and leave a multicast group, respectively.

the calculated rate differ and when the data rate does not follow the calculated rate.

Although the equation-based approach of TFRC has got considerable attention in the research community in the past years, it has not been studied in detail under what conditions equation-based rate estimation is applicable to layered multicast scenarios where the sender emits data to layers at very slowly varying rates defined by the application.

1.2 Contribution

We designed and conducted a large set of experiments both in NS-2[3] simulator and in a clinical laboratory testbed running FreeBSD systems to investigate the equation-based approach applicability to open-loop layered multicast congestion control. As an instance of equation-based algorithm we used TFRC, state-of-the-art equation-based congestion control protocol. Previous simulation studies have suggested that the TFRC rate calculation depends on the sending rate [RSS03]. Floyd et al. indirectly note in [FHPW00a] that such a dependency indeed exists:

> For a fixed loss probability, the faster the sender transmits, the lower the loss event fraction. However, the sending rate is determined by the congestion control scheme and so itself depends on loss event fraction.

The quote holds for a closed-loop unicast scenario. However, in an open-loop receiver-driven multicast scenario the sending rate is not controlled by the congestion control scheme but by the application defined sending rate. We show in detail how the computed loss event rate depends on the packet sending rate in the open-loop scenario. More precisely, the loss event rate computation of the TFRC algorithm computes lower loss event rate for a steady loss process as the number of packets per round-trip time increases. The problem originates from

[3]NS-2 is available under http://www.isi.edu/nsnam/ns/index.html

the aggregation of multiple losses in a round-trip time to one loss event. The aggregation depends directly on the packet rate.

TFRC has been designed to operate with packets of fairly constant size by varying the packet rate. However, throughput can be controlled also by altering the packet sizes, not just the packet sending rate. This is indeed more suitable for many multimedia applications that are inherently packet rate based and may easily modify the packet size. During our work we became aware of a contribution of Widmer et al. that modified the TFRC loss event rate algorithm to better cope with variable packet sizes [WBB04]. We show that this modification does not solve the loss measurement bias present in the open-loop control.

As the current TFRC and its modified versions do not directly adapt to the open-loop multicast scenario we propose a modification to the loss event rate algorithm. We modify the loss event rate computation to compute true packet loss rate over a history and show that the computation is independent of the sending rate and is better applicable to the open-loop multicast scenario.

The modified loss estimator allows diverse approaches to subscription level management at receivers. We discuss some possible subscription level management strategies to illustrate the potential of the equation-based approach to layered multicast congestion control. We also implement one of the strategies and, using simulation, show that it performs well over a wide range of network conditions. It manages to avoid frequent layer oscillations but claims its share of the bandwidth without sacrificing TCP-compatibility.

1.3 Outline

This thesis is structured as follows: in Section 2 we discuss background issues, such as congestion control and multicasting in general, and form the basic knowledge required in the subsequent Sections. In Section 3 we discuss related work, including TCP and several multicast congestion control algorithms. In Section

4 we describe how we analyzed the TFRC algorithm applicability to open-loop scenario and discuss the results showing the dependency on the sending rate. In Section 5 we present our modification to the TFRC loss estimation algorithm and show simulation results of the modification. In Section 6 we discuss possible subscription level management strategies. We also describe our implementation of one management strategy and show, using simulation, that it performs well over a wide range of network conditions. In Section 7 we conclude the thesis and summarize the work and discuss open issues.

2 Background

In this section, we provide an overview of congestion control and multicasting. We start by discussing the congestion control in the Internet. We then describe two group communication methods, broadcasting and multicasting, and their difference and we finish the section with a discussion about requirements for layered multicast congestion control.

2.1 Basics of Congestion Control

It is much due to the TCP congestion control algorithms that the current Internet still operates so well. Internet, as a congestion control system, is an instance of a general control system, or a feedback system. Internet is one of the largest deployed artificial feedback systems in the world. As it continues to expand in size and applications it becomes increasingly important to understand the principles of congestion control and to deploy congestion control algorithms in new protocols operating in the public Internet.

2.1.1 Threat of Congestion Collapse

In circuit switched network, e.g., in a traditional Public Switched Telephone Network (PSTN), a dedicated connection is reserved for the communicating end-systems. Such a mechanism can guarantee a certain level of quality (meaning mostly bandwidth) but is somewhat poor use of resources. In quiet periods, when one is speaking and the other listening, or even worse, when both are quiet, the end-to-end path is still reserved but effectively unused.

Internet is a best-effort packet switched data network that does not guarantee successful delivery of packets to destinations. That is, it does its best to do so, but does not guarantee it. Internet operates on a protocol called Internet Protocol (IP) [Pos81a] which is a packet based protocol designed for end-to-end communication between hosts (a.k.a. end-systems, or ES) in the Internet. In the network IP packets are forwarded by entities called routers. A router temporarily stores IP packets in a buffer and forwards them as quickly as possible. When a router has too many IP packets in the buffer, some of them may be dropped. This is equivalent to congestion.

While the Internet, and a packet switched network in general, reaches a higher utilization through more efficient use of the "quiet periods", it also introduces problems not present in a circuit switched network. One of the greatest challenges is congestion control.

Network congestion occurs in routers when the incoming data arrival rate grows beyond the output link capacity. Uncontrolled traffic in a congested network may lead to a state called *congestion collapse* where practically no useful data gets through the network [FF99]. To keep the network operational in presence of congested links, sources must lower the sending rate to the level of the network capacity, or below it.

2.1.2 Equilibrium and Stability

In congestion control, end-systems respond to congestion by lowering the sending rate, and by attempting to find a new *equilibrium* point. Equilibrium point is control theoretical jargon. In the context of network congestion control, it means that sources adjust their sending rates to levels where all consider the rate right for the given congestion level.

When a protocol fails to find an equilibrium point, or the found equilibrium is unstable, the protocol is said to *oscillate*. An oscillating system does not utilize resources optimally and is generally considered bad. For a unicast flow, oscillation means variations in sending rate. The more the sending rate varies, the more the protocol oscillates. For a layered multicast congestion control oscillation means the frequency of changing the subscription level (subscription level is defined in Section 2.3.1). The more there are changes, the more the protocol oscillates.

The equilibrium point depends on many factors, such as network topology and transient sporadic traffic crossing a bottleneck link. Assuming an algorithm finds an equilibrium point for a given congestion level, it is of great importance, how quickly it finds it, or how quickly it *converges* to the equilibrium. The longer it takes to find the equilibrium, the more the network is either underutilized or overloaded. Different algorithms have different properties in terms of equilibrium, stability and reactiveness. For more insight to equilibrium and stability we refer the interested reader to [LPD02].

2.1.3 Insufficient TCP

TCP has incorporated congestion control algorithms since a series of events in mid 80's that became to be called the Internet meltdown, technically called congestion collapse [Jac88]. Since then a large body of research has been published on congestion control. There is wide consensus in the research community that congestion control is required also from future protocol operating in the Internet

[BB98].

Although TCP has performed above the expectations, it has been accepted not to be sufficient. In [BB98] this is expressed in clear vocabulary:

> It has become clear that the TCP congestion avoidance mechanisms, while necessary and powerful, are not sufficient to provide good service in all circumstances.

The way Internet is being used now, compared to the way it was used 20 years ago, is radically different. Since then Internet has grown in topology, traffic and applications and it has become a de facto standard of communication for research communities as well as for many small and large businesses. Its role as an important resource has become obvious.

The change with strongest impact is due to the invention of World Wide Web (WWW). WWW runs on Hypertext Transfer Protocol (HTTP) [FGM+99]. Typical to WWW page construction is to build a complete page from small fragments, such as images. HTTP uses TCP connections to fetch page components. The connections are often short, only a few packets. Such short TCP flows are next to impossible to congestion control in the Internet. Many short HTTP connection never last long enough to use the congestion avoidance algorithm and thus do not really adapt the traffic to the congestion level of the network.

WWW traffic plays a significant role in packet level measurements. Short packets of HTTP protocol are reported to be in dominant role. However, fat connections, such as FTP downloads, appear to dominate the bandwidth consumption. An interesting observation made in [FML+03] is that peer-to-peer and streaming traffic are strongly increasing. WWW made the Internet traffic more heterogeneous. The trend of peer-to-peer and streaming may well boost the impact.

2.1.4 Active Queue Management

Congestion control is performed at end-systems by adjusting the data sending rate to accommodate the congestion state of the network. To properly adjust the sending rate a source must have a good estimate of the level of network congestion. End systems estimate the level of congestion on their path by detecting congestion signals. Theoretically, the more congestion in the network, the stronger congestion signal should be detected. The responsibility of congestion signaling is at routers, i.e., in the network components on an end-to-end path. An explicit congestion signal can be a packet loss (explicitly dropped by a router) or a marked packet (marked by a router instead of dropping).

The simplest queue management algorithm is Drop-tail. It simply drops packets when they no more fit in the output queue of a router interface. Drop-tail is easy to implement: it is inherently present in all algorithms in the form of finite buffer space. The unfortunate property of Drop-tail algorithm is that it signals congestion only after it has occurred. The implicit signal in a Drop-tail algorithm is queuing delay: in congestion, as packets arrive at higher rate they can be sent, the queue starts to build up. It follows that packets must wait in the queue and the round-trip time increases. However, increased delay means also loss of performance.

Active Queue Management (AQM) algorithms have been developed to overcome the late signaling of congestion present in the Drop-tail algorithm. The most widely cited and implemented AQM algorithm is Random Early Detection (RED) [FJ93]. It computes an average queue length using a Exponential Weighted Moving Average (EWMA) of queue length samples. The congestion signaling, i.e., packet dropping or marking, is an increasing function of the average queue length. Up to a threshold th_{min} the drop probability is zero, between $th_{min} - th_{max}$ the drop probability increases linearly from zero to p_{max}. When the average queue length is above th_{max}, all packets are dropped. Thus, as the average queue increases, so does the dropping or marking probability.

Random Exponential Marking (REM) [ALLY01] is an another AQM algorithm that decouples congestion measure and performance. REM measures congestion using a variable called *price* and marks (or drops) packets using an exponential probability function of the price. Thus, the higher price, the higher probability of marking. The novel idea is that the price is *increased* or *decreased* using the average queue length. If the length is beyond a small target length, then the price is incremented. If the length is below the target length, then the price is decremented. This allows the algorithm to have a high price and high marking probability with short queues. For RED to have a high marking probability, a long average queue is required. Thus, RED implicitly introduces loss of performance (as queuing delay).

Above, *marking* assumes an Internet Protocol feature called Explicit Congestion Notification (ECN) [RFB01]. ECN is designed to allow routers to notify of congestion without dropping packets but by raising a flag bit in headers of packets. Sources understanding the flag bit react to the bit as if the packet was lost. This approach does not waste network resources as much as explicitly dropping packets.

2.1.5 Network Utility Optimization

In principle, Internet is a shared resource and end-systems are users of the resource. Such a complex system can be described in a number of ways. In [LPD02] Low et al. model it as a control system and translate the language of control theory into a common language of commerce.

In the terms of commerce the routers in the network are offering services, which translates to bandwidth. Each router has a limited capacity of service to offer, that is, a limited bandwidth. End systems are consumers on the market purchasing services from routers, which translates to using the offered capacity.

In the theoretical model the routers sell the resources for zero price until the demand grows beyond the offer. That is, when sources send more data than there

is capacity in a router, the affected router starts increasing the service price. The concept of price translates to congestion signal. The higher the price, the greater the congestion in the network and the stronger the congestion signal should be provided.

Sources are modeled, as often in commercial optimization models, through utility functions. A utility function represents source's wealthiness. The higher utility, the bolder the source; it follows that a source naturally wants to grow its utility. However, a utility function is constructed in such a way that it guarantees decreasing gain in utility as the price increases. It follows that at high prices a major investment is required even for a small increase in utility. It leads to the concept of optimization.

A source wants to optimize the price/service performance and that is analogous to optimally using the shared network. Congestion control algorithms should globally reach optimal use of the shared network, including fair competition over the resource.

2.2 Group Communication

Group communication means communication between many participants such that the communicating systems are in a logical relationship with each other. While group communication may be implemented over unicast connections between all group members, we limit our discussion to native group communication data delivery mechanisms. In group communication the network is virtually aware of the group existence and offers automatic message delivery to group members. Broadcast is the simplest method of group communication. Multicast is more complex but offers scalability unavailable in broadcasting.

2.2.1 Broadcast

Computer networks are constructed using a layered architecture. The Internet architecture has five layers (see Section 1.7.2 in [KR00]). The lowest layer is *Physical Layer* and it covers the physical mediums. Physical mediums are, among others, twisted-pair copper wire and optical fiber. The second layer is *Link Layer*. It is covers how bits are encoded to a medium. A popular link protocol is Ethernet.

On the link level the medium may be directly or indirectly shared among hosts in the same local area network (LAN). That is, when a host sends a message over the medium, all other hosts see the message. Using addressing schemes hosts know whether the message is destined to them. Link protocols, such as Ethernet, have a specific broadcast addresses. All hosts catch the messages destined to the broadcast address. This way, a message that is sent to all hosts in the LAN, is efficiently delivered as a Link Layer broadcast message.

In the Internet architecture, the third layer is *Network Layer*. The standard protocol on the Network Layer is the Internet Protocol (IP) [Pos81a]. It provides an addressing structure which includes a broadcast address per configured logical network (IP-network). Sending a message to the IP broadcast address effectively delivers the message to all hosts in that IP network.

When the IP network is small, the IP broadcast resembles Link Layer broadcast. However, an IP network may be extremely large for a large organization. Sending an IP broadcast message in such a network may flood to thousands of recipients, even though only a fraction of them had interest in the message.

Broadcast is a practical group communication mechanism for local area networks where messages are sent only occasionally, for a relatively small set of recipients. For larger networks it has severe scalability problems and even in the IP addressing architecture broadcast messages may not be sent outside the organization IP-network. Thus, broadcasting is inherently not scalable.

2.2.2 IP Multicast

IP Multicast (or, simply multicast) is an advanced group communication mechanism designed to operate on the Internet Protocol (IP) and User Datagram Protocol (UDP) [Pos81a, Pos80]. In the Internet, the network components that are responsible of forwarding packets closer to their destinations are called routers. Normal IP packets contain a source and destination address and routers use the destination address to forward the packet in right direction. An IP (version 4, or IPv4) address is usually given in dotted decimal notation, e.g. 130.83.139.101, but routers handle them in a compact 32 bit format.

In multicasting, the addresses are interpreted in a different way. By definition, a multicast packet may have multiple recipients, all of which are impossible to address in an IP packet. Instead of stamping packets to specific recipients, the packets are destined to a *multicast group*, independent of recipients. It is the network's responsibility to forward the packet appropriately.

A host wanting to receive packets sent to a specific multicast group must subscribe to the group first. The network, i.e., the routers in the network, then becomes aware of the membership and starts to deliver the group packet to the new member. All of the complexity of packet delivery to group members is in the network.

The novelty in multicasting is in the group paradigm. Theoretically, any host in the Internet may be a sender, and any host in the Internet may subscribe to a multicast group. When a packet is sent to the group, all members of the group receive the packet, independent of the physical (compare to LAN broadcast) or logical location (compare to IP broadcast). The multicast routers replicate the packet only once per link. If there are $N > 1$ recipients behind a link, then unicast communication would require N packets with the same content to be sent explicitly to each of the N recipients. Multicast requires the packet to be forwarded only once.

2.2.3 Group Management and Routing

The basic operation of a multicast router (a router that, in addition to normal routing, performs multicast routing) is theoretically quite simple. When a packet destined to a multicast group g is received from interface i_{src}, the router forwards the packet to all interfaces $i \neq i_{src}$ that have subscribed members.

The challenge of multicast routing is in the group management. Multicast routers must keep track of memberships such that they can any time make decisions whether to forward packets of group g to an interface $i \neq i_{src}$. The more dynamic a group is, the more routers must interact to share information on group memberships.

A host joins a multicast group by sending an IGMP join message to the nearest router. IGMP stands for Internet Group Management Protocol [CDK+02]. This way the router becomes aware of the host's desire to receive packets destined to the joined multicast group. If the router already receives the packets of the joined group, then it does not have to anything. However, if the host issues the first join for the group, the router must propagate the request for the group. The request cascades up in the routing tree such that the multicast routing algorithm becomes aware of the new participant.

The routing algorithm, centralized or distributed, saves some state of the group memberships. However, a group member can also simply become uninterested in the group by, for example, terminating the receiving application. An explicit IGMP leave message can be sent to quit the group so that the routing algorithm may update the group state accordingly. The delay between issuing the leave request and when the data actually stops is called *leave latency* [CDK+02].

There are situations where a flow should be ceased. Such a situation is, for example, when a computer crashes. There is little use for the transmitted data when the receiving computer has vanished from the network. Therefor the multicast routing algorithm implements soft states that must be refreshed in certain inter-

vals. Without refreshing, the information is considered aged and actions are taken as if a leave message had been received. For more information about multicast routing we refer the interested reader to [Ram00].

2.3 Requirements for Layered Multicast

In this section, we present requirements for layered multicast congestion control algorithms. We start by defining layered multicast and proceed with the requirements. We identify and discuss the following aspects: fairness, aggressiveness and responsiveness, smoothness, scalability and independency.

2.3.1 Layered Data Organization

In a layered multicast session a source emits data to a set of cumulative multicast groups called layers. Each layer carries data with bandwidth L_i in such a way that the cumulative data rate on layer i is

$$B_i = \sum_{j=0}^{i} L_j$$

We call this *cumulative layered data organization*. In cumulative layered data organization the sender must emit data such that the cumulative data carried on layer i is a subset of cumulative data carried on layer j for all $i < j$. That is, the base layer carries the minimum set of data to deliver. The second layer delivers additional data that may be decoded in conjunction with the base layer. The third layer provides additional data, and so on. All layers together offer the maximum set of data to deliver.

In layered multicast congestion control approach the congestion control is performed by receivers through subscription level management. In cumulative layered data organization the order in which layers are subscribed is predefined. Sometimes layered multicast congestion control is called multi-rate congestion

control. We use the term layered and interpret it as an instance of general multirate, where there is possibly no predefined order of subscription.

2.3.2 Fairness

Fairness if often important when sharing resources. In operating systems many resources, such as CPU, are shared among users. Network is also a shared resource that is shared by users. Fair competition over resources mean that all get an equal share of the resource. If one participant grabs more resources than others, it is considered unfair and it discriminates other participants.

Fairness of an algorithm over a shared resource can be measured using a fairness index as follows

$$f(x) = \frac{(\sum x_i)^2}{n \sum x_i^2}$$

where x_i is a share of participant i and n is the number of competitors. When all get equal share of the resource (i.e., $\forall i \neq j : x_i = x_j$), the fairness index is 1 [JCH84]. The more there is unfairness, the closer to $1/n$ the fairness index approaches.

Congestion control algorithms can be considered competing over a shared resource, i.e., compete over bandwidth. We are particularly interested in fairness to TCP which is the dominant protocol in the Internet. Practically all new protocols will have to coexist with it. For this reason it is important not to discriminate it. Fairness to TCP is also referred to as TCP-compatibility which is defined in Section 3.1.5.

2.3.3 Reactivity

There are basically two questions a layered multicast congestion control algorithm must provide an answer to: 1) whether the maximum subscribed layer should be unsubscribed; or 2) whether the next layer should be subscribed. Subscribing less layers than the network capacity would allow is poor use of re-

sources. Subscribing too many layers causes TCP-incompatibility, congestion and is unsocial behavior.

Generally, on spare capacity, bandwidth consumption should be increased. Similarly, on congestion, bandwidth consumption should be decreased. In the context of receiver-driven layered multicast, this process is driven by subscription level management. Depending on the current equilibrium rate, a proper subset of layers should be subscribed. When the equilibrium rate changes, the subscription level is altered, if necessary.

Reactivity is the speed of reacting to changes in the equilibrium rate. Reactivity can be split into *aggressiveness* and *reactiveness*. Aggressiveness means the speed of claiming spare bandwidth and reactiveness means the speed of giving up bandwidth in case of congestion. TCP implements aggressiveness through growing congestion window with one packet per round-trip time and reactiveness through halving the congestion window after detecting congestion. A layered multicast protocol implements aggressiveness and reactiveness by altering the subscription level.

Optimally, a protocol is reactive, claims its fair share and competes fairly. However, in multicasting this poses a challenge. Being reactive means executing frequent layer changes. Each layer change is a fairly expensive operation. As layers may be coarse grained, it potentially introduces heavy oscillations. Coarse grained layers make it also hard to claim the fair share when the equilibrium rate is in the middle of two layers. Thus, in the context of layered multicast congestion control trade-offs need to be done between reactivity, oscillations and fairness.

2.3.4 Scalability

In addition to optimal layer subscription a multicast congestion control algorithm should be scalable. Implementing a closed control loop generates feedback messages from receivers to the sender. Feedback has two faces. First, such an approach is prone to *feedback implosion*, meaning that the traffic is dominated by

feedback rather than data, and must be avoided. Secondly, if the protocol performance depends on the feedback frequency from all receivers, the session size has an upper bound.

Many multicast congestion control algorithms nevertheless require feedback. To avoid feedback implosion the feedback must be controlled in a scalable manner. Several approaches to avoid feedback implosion have been studied in the literature, e.g., exponentially distributed timers [NB98].

The state information required to execute the congestion control should not grow linearly with the session size. For large session, e.g., with tens of thousands of participants, the state information would eventually grow beyond capacity. Thus, exact tracking of session size becomes unpractical for large sessions and when the session size is required, it should be estimated by using other mechanisms than exact tracking.

2.3.5 Independence

Relying too much on network features other than packet forwarding, such as performance of operations on multicast routing, couples the congestion control to systems that are hard, if not impossible, to control. Thus, a scalable multicast congestion control algorithm only expects basic multicast packet forwarding and conservative, in terms of performance, routing services.

A bad subscription decision in a multicast session typically has a significantly stronger negative impact than in unicast. This is due to *leave latency* (see Section 2.2.3). The leave latency is the time between when the last member stops listening to a source or group and when the traffic stops flowing [CDK+02]. This time can be anything from milliseconds to order of seconds. Hence, a multicast congestion control protocol should avoid bad subscriptions more than a comparable unicast session of overestimation.

3 Related Work

In this section we discuss related work. We describe TCP congestion control, steady state TCP-models and four multicast algorithms: Receiver-Driven Layered Multicast (RLM), Receiver-Driven Layered Congestion Control (RLC), TCP-Friendly Multicast Congestion Control (TFMCC), and Smooth Multirate Multicast Congestion Control (SMCC).

3.1 Transmission Control Protocol

Transmission Control Protocol (TCP) [Pos81b] is a reliable connection-oriented full-duplex end-to-end unicast protocol. It guarantees that data sent from a source application is delivered to the receiving application in the original order and free of errors. It expects best-effort delivery from the protocol layer below, usually the Internet Protocol (IP) [Pos81a], with no guarantees beyond best-effort delivery.

The communicating ends must establish a connection before any data may be sent and they must keep state for the connection during the existence of the connection. The established connection may be used to send data in both directions (full-duplex).

TCP is a stream protocol meaning that an application using TCP service provides a stream of bytes instead of packets (a.k.a. segments). TCP is responsible of splitting the byte stream into packets and to reconstruct the byte stream from packets at reception. Thus, although an application sends bytes, TCP itself carries the data in packets. Splitting the outgoing byte stream into packets involves allocating increasing sequence numbers for the packets. The sequence numbers indicate the original order of packets and the data they carry.

3.1.1 Self-Clocking

A TCP receiver acknowledges all correctly received packets, i.e., packets that arrive free of errors in the order of the sequence numbers. A packet that arrives *out of order* is reordered to guarantee the order property. A packet that has been corrupted on the way is discarded as if it had never arrived.

TCP sender uses receiver's acknowledgments to *clock* the sending of new packets. Each acknowledgment informs the sender that a packet has left the network and that a new packet may be injected in the network. The loop of sending packets and receiving acknowledgments to drive sending is called *Self-Clocking* [Jac88]. Self-Clocking effectively prevents a sender transmitting packets faster than packets leave the network, or the receiver can process them.

The time it takes to receive an acknowledgment to a packet is called *round-trip-time* (t_{RTT}). To fully utilize the available bandwidth of a network path a TCP sender should have W packets of size s_i in transmission where the sum of data in the packets $\sum_{i=1}^{W} s_i$ equals to the *bandwidth delay product*

$$B = C \cdot t_{RTT} \tag{1}$$

where C is the end-to-end path available bandwidth.

3.1.2 Slow Start and Congestion Avoidance

TCP implements a window-based congestion control protocol where the receiver acknowledges all correctly received packets. The higher the bandwidth delay product the more packets must be kept in transmission to fully utilize the available bandwidth. The maximum number of packets in transmission (i.e., unacknowledged packets) is limited by (1) receiver advertised flow control window ($fwin$), which is effectively the free space in receiver's input buffer, and (2) congestion window ($cwin$) maintained by the sender.

TCP congestion control implements two algorithms: *Slow-Start* and *Congestion Avoidance*. The protocol starts with slow-start and sets the *cwin* to 1 packet. In

slow-start the congestion window is doubled every round-trip time (or incremented for each acknowledgment) until a threshold size. Crossing the threshold makes the sender switch to the congestion avoidance algorithm. Slow-start is used to start the self-clocking algorithm used in congestion avoidance: self-clocking requires data flowing to operate but to send data self-clocking is required [Jac88].

In congestion avoidance, the congestion window is incremented every round-trip time (or with $1/cwin$ for each acknowledgment). Hence, the congestion window grows exponentially in slow-start and linearly in congestion avoidance.

3.1.3 Detecting and Recovering from Packet Loss

The responsibility of detecting a packet loss is at the sender side. In principle, a missing acknowledgment is a signal for a packet loss. The only reliable mechanism to detect a missing acknowledgment is a timeout timer.

Basic TCP detects a lost packet through a timeout timer. When an acknowledgment for a packet has not arrived and the timeout timer fires the source considers the packet lost. Upon detection a packet loss, the congestion avoidance threshold is set to half of the current congestion window size, the congestion window size is set to one packet, and the source switches to the slow-start algorithm. Then the lost packet is retransmitted [Jac88]. Increasing the congestion window linearly and reacting to congestion by halving the congestion window is often referred to as *Additive Increase Multiplicative Decrease (AIMD)*.

An algorithm called *fast retransmit* improves TCP behavior in case of packet losses. Fast retransmit avoids waiting for the timer to fire before retransmitting the lost packet. A receiver sends a duplicate acknowledgment for the last packet arrived *in order* for each packet received *out of order*. Fast retransmit is executed when three duplicate acknowledgments have been received for a packet S_i. The sender then retransmits packet S_{i+1} [APS99].

Another algorithm called *fast recovery*, implemented in conjunction with fast retransmit, effectively avoids entering slow-start state after fast recovery. It improves TCP speed of recovery after fast retransmit by halving the congestion avoidance threshold value and setting the congestion window size to the threshold value. It then continues in congestion avoidance state with the new congestion window size [APS99].

TCP has been extended with options that enable more efficient congestion control. One such extension option is Selective Acknowledgment (SACK) [MMFR96]. Basic TCP is able to indicate packet loss and the particular packet that has been received in order. SACK introduces a mechanism to acknowledge ranges of packets. Using SACK option a source can learn in one round-trip time exactly what packets have successfully arrived to the destination and retransmits only the lost fragments. This limits the need for retransmissions to only those packets that have been lost on the path.

3.1.4 Analytical Models of TCP

Long-term steady-state throughput of TCP depends on factors such as packet loss rate, round-trip time and retransmission timeout. TCP long-term steady-state behavior can be modeled using simplified models such as given in [MSM97]. A simple TCP throughput equation is given in Equation 2

$$T = \frac{s \cdot c}{t_{RTT} \cdot \sqrt{p}} \tag{2}$$

where s is the packet size, c is a constant (commonly set to $\sqrt{3/2}$), t_{RTT} is the round-trip time and p is the packet loss rate.

At high packet loss rates the TCP retransmission timeout plays a more important role. A more complex model that takes the TCP retransmission timeout into account is studied in [PFTK98] and given in Equation 3

$$T = \frac{s}{t_{RTT}\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}}) \cdot p(1 + 32p^2)} \tag{3}$$

where s is the packet size, t_{RTT} is the round-trip time, t_{RTO} is the retransmission timeout and p is packet loss probability.

The model of Equation 3 divides time into blocks called *rounds*. Each round equals to the current round-trip time. A round i starts with sending W_i back-to-back packets and ends when the first acknowledgment for one of the packets is received, i.e., after a round-trip time. Packet losses in *different* rounds are assumed to be uncorrelated. Packet losses in the *same* round are assumed to be correlated and the packets after a lost packet are considered lost, too. Physically lost, as well as packets only considered lost by the model, constitute the packet loss rate p of the model.

The assumption of packet loss correlation of the back-to-back packets is based on the assumption that the links on the end-to-end path implement Drop-tail queuing policy. In such links packet drops are caused by buffer overflow. The buffer is likely to remain full for all back-to-back packets following the initial packet loss. However, in links implementing Random Early Detection [FJ93], packets are dropped *before* the queue becomes full according to a probability function of the average queue length. The correlation between losses of back-to-back packets in a round does not hold in presence of RED links. In fact, the assumption does not hold for any Active Queue Management algorithm that drops packets according to a probability function.

3.1.5 TCP-Compatibility

In the Internet TCP is the dominant protocol. According to packet-level measurements more than 90% of traffic is TCP [FML+03]. This suggests that any protocol operating in the public Internet will coexist with TCP. The dominant role and wide range of applications running on TCP encourages to develop TCP-compatible protocols instead of radically changing TCP itself.

A TCP-compatible flow consumes no more bandwidth than a TCP flow running under comparable conditions [BB98]. The conditions include round-trip time,

end-to-end path MTU, packet loss rate, packet size etc. The definition of TCP-compatibility provides us the basis for compatibility. Low writes in [Low03] about fairness that

> ... fairness of TCP algorithms should not be defined solely in terms whether they receive the same equilibrium rates, as commonly done in the literature, because the equilibrium bandwidth allocation generally also depends on AQM, network topology, and routing etc.

and shows how different flavors of TCP algorithms achieve significantly different throughputs under different conditions. This makes it hard to define TCP-compatibility. In our simulations we have chosen the TCP/Sack implementation of NS-2 simulator and analyze fairness to it.

From Equation 3 it can be seen that TCP throughput depends linearly on the packet size. However, a multimedia flow that is bound to some application defined packet sizes should be entitled to a fair share of the network capacity independent of the forced packet size. We relax the TCP-compatibility paradigm to consider flows sending small packets TCP-compatible as long as the small packet throughput is smaller than, or equal to, a TCP flow on the same path.

3.1.6 TCP-Friendly Rate Control

TCP-Friendly Rate Control (TFRC) is an attempt to provide a congestion control algorithm for unicast multimedia flows requiring a smooth sending rate without compromising TCP-compatibility [FHPW00b]. We give a short overview of TFRC here and describe it in more detail in Section 4.2.

TFRC uses the analytical model of TCP presented in Equation 3 to compute a TCP-compatible sending rate. To do the computation, TFRC requires all of the input parameters to the equation, i.e., round-trip time, retransmission timeout, packet size, and packet loss rate.

TFRC estimates round-trip time by frequent feedback messages from the receiver. Single round-trip time measurements are filtered to make a smoother estimate of the round-trip time. Retransmission timeout is directly derived from a round-trip time measurement. Packet size is the size of packets or mean size of packets. The protocol has been designed to work with fairly constant packet size, i.e., size of packets in a flow should not vary much.

TCP increases the congestion window until a loss event (one or more packets lost in a window of packets, a.k.a. loss event), and it *reacts* to a loss event by halving the congestion window. This produces an oscillating throughput. TFRC does not directly react to packet losses. Instead, it *estimates* the loss event rate and feeds the estimate in the equation as packet loss rate. In presence of a steady loss event rate the estimate does not change and the equation computes a smooth sending rate.

TCP does not need any history of loss events because it simply reacts to them when detected. However, TFRC requires knowledge of the near history to make the estimate. Without a history it would be hard to know whether a loss event changes the loss event rate or not. The history is bound to a number of loss events which makes the history dynamic. If loss events are experienced frequently, the history is adjusted short. If loss events are experience rarely, the history is adjusted long. When the history is short, it means that there has been congestion in a short period of time. When the history is long, it means that there has not been much congestion recently.

These estimated values are fed in the equation to produce a TCP-compatible sending rate. TFRC directly adjusts the sending rate to the value given by the equation.

3.2 Multicast Congestion Control Algorithms

Communication between two hosts, such as with TCP, is called unicast. Group communication between many end-systems may be implemented using unicast communication between participants, broadcasting or multicasting. Using unicast to implement group communication is not scalable. Broadcasting is limited to a local network and does not scale to larger networks. Multicasting spans theoretically across the whole Internet and is the only scalable solution to group communication.

In multicasting the network, i.e., routers in the network, is responsible of replicating data packets to their destinations. Multicast groups are either *one-to-many* or *many-to-many* groups. In the first a single source sends data to the multicast group to many receivers, e.g., an Internet radio. In the latter many sources send to the same group and all group participants receive the data, e.g., a conference call.

The activity of data packet replication, including managing multicast group memberships, is called multicast routing (see Section 2.2.3). Performance of multicast routing algorithms typically depend on group size, group dynamicity and whether the group is one-to-many or many-to-many, latter being more complex [Ram00].

Multicast congestion control is the task of a multicast sender and receivers to adjust the multicast traffic in the end-to-end path between the sender and a particular receiver according to the measured congestion. Different receivers in a multicast group do not necessarily share same conditions; in fact, two group members don't necessarily share even a single link on their path to the sender. These kind of differences in a multicast group is called *heterogeneity*.

Multicast congestion control algorithms can be classified roughly into two categories: single-rate and multi-rate protocols. Single-rate protocols are mostly sender-driven: the sender adjust the transmission rate to match the slowest re-

ceiver in the group. In multi-rate protocols the sender emits data to several multicast groups, usually, independent of the receivers and expect receivers to subscribe a subset of groups depending on the end-to-end path between the sender and the particular receiver.

3.2.1 Receiver-Driven Layered Multicast

Receiver-Driven Layered Multicast (RLM) [MJV96] is the pioneer work of receiver-driven layered multicast congestion control. It introduced the concept of splitting a multicast session into multiple layers and having a passive sender taking no active role in the congestion control. It merely emits data to the layers. Congestion control is carried out by receivers by searching for an optimal *level of subscription* by joining and leaving layers.

In principle, RLM receivers run a simple loop: on congestion, leave the highest layer, and on spare capacity, join the next layer. When a receiver has a too high level of subscription, it causes congestion that can be detected in the received data flow as increased packet loss rate. Detecting spare capacity is not as easy. RLM infers the spare capacity by periodically performing timer-triggered *join-experiments*. A join-experiment begins by the receiver advertising its join-experiment in a multicast group. The notification allows "shared learning" since other receivers may detect failed join-experiments without doing join-experiments themselves. Shared learning is crucial to the scalability of RLM. However, through shared learning other receivers may only learn what does not work - not what does work. Unless shared learning indicates failure of a join-experiment of a fellow group member, the only way to learn is to perform a join-experiment.

RLM discovered the top of the iceberg and lacks some desired properties. There are scenarios where RLM provides poor fairness to different flows as well as to other RLM flows. Legout shows in [LB00] how the convergence speed and packet loss rate suffer from small layer granularity. Despite its shortcomings, RLM inspired the community to study receiver-driven approach producing significant

advances in the field.

3.2.2 Receiver-Driven Layered Congestion Control

Receiver-Driven Layered Congestion Control (RLC) [VCR98] is a fully distributed congestion control algorithm that requires no communication between receivers nor feedback from receivers to the sender. The total lack of feedback offers virtually unlimited scalability. RLC elaborates ideas of RLM, such as join-experiments and shared learning.

RLC achieves near TCP-compatibility by imitating TCP behavior through joining and leaving exponentially distributed layers such that joining a layer duplicates the cumulative rate and leaving a rate halves the cumulative rate. Congestion avoidance and linear increase is implemented by exponentially distributing the layers rates such that base layer rate is $L_0 = B_0$ and

$$\forall i > 0 : L_i = B_i - B_{i-1}$$

and by exponentially distributing the join timer intervals on different layers

$$t_i = t_0 \cdot 2^i$$

where t_0 is the join timer interval on the base layer. When TCP does a window-halving, RLC imitates it by leaving the highest subscribed layer and thus halving the rate.

Join-experimenting and shared learning in RLC is implemented using special synchronization packets (SP) emitted by the sender to layers L_i at time intervals t_i. That is, only when a receiver detects an SP packet on the highest subscribed layer, the receiver may perform a join-experiment to the next layer. Since all receivers receive the SP packet at almost same time, they will perform the join-experiments simultaneously. This is crucial in scenarios where multiple receivers are behind the same bottleneck link. Using SP's and shared learning they all experiment and learn simultaneously the failed join-experiment.

RLC introduces sender initiated bandwidth probes to assist bandwidth inference. The sender periodically emits bursts of packets to one layer at a time such that the bandwidth of the layer is doubled. This is expected to increase queue occupancy and to cause packet losses that indicate the receivers not to increase the subscription level.

We consider RLC the state of the art layered multicast congestion control protocol. The simplicity compared to what it achieves is respectable. The total independence of feedback makes the protocol free of feedback implosion threat.

However, RLC does not deliver everything. Its TCP-compatibility bases much on the exponential distribution of layers. This is applicable in scenarios where the layers may be fixed and the application may adjust data to layers. For multimedia data this is not always practical. Also, at higher layers the fixed granularity of the layer rates becomes so large, that it is not always easy to utilize efficiently.

3.2.3 TCP-Friendly Multicast Congestion Control

The development of long-term steady state TCP-models with the definition of TCP-compatibility boosted the search for alternative congestion control protocols that could peacefully coexist with TCP. An outcome of the efforts is TCP-Friendly Rate Control (TFRC), an equation-based congestion control protocol for unicast flows [FHPW00b].

TCP-Friendly Multicast Congestion Control (TFMCC) [WH01] is a single-rate equation-based multicast congestion control protocol. It is an extension of TFRC to multicast scenario. Being a single-rate protocol, TFMCC is sender-driven and adjusts the sending rate to the slowest receiver in the session. The slowest receiver is called Current Limiting Receiver (CLR). At any given time there may be only one CLR, but over a time span, the CLR may vary. The sender injects in all data packets the id of the CLR. When the CLR receives a data packet with its id, it becomes aware of being the CLR and send feedback to the sender more

frequently than non-CLR members in the group. Hence, the sender and the CLR form a closed-loop control.

The underlying TCP-model requires estimates of packet loss rate, round-trip time, TCP retransmission timeout. The protocol focuses much on feedback suppression and accurate parameter estimation of the CLR. However, as non-CLR members of the group form a near-open-loop control with the sender, their estimate of a TCP-compatible rate may not be accurate. Also, being single-rate protocol, it scales poorly in presence of heterogeneous set of receivers.

3.2.4 Smooth Multirate Multicast Congestion Control

Smooth Multirate Multicast Congestion Control (SMCC) [KB03] is an interesting attempt to merge multi-rate and single-rate approaches. Although SMCC is a multi-rate protocol, it is not really receiver-driven. Instead, it employs TFMCC on each layer and implements complex mechanisms to control receivers' join and leave decisions as well as the join process.

Each layer has a Current Limiting Receiver (CLR) as in TFMCC. The sender forms a near-closed-loop with the CLR of each layer. The layer rate is adjusted by the sender to the CLR rate. When a CLR estimates a rate below the layer minimum limit, the CLR leaves the layer. The layer then seeks a new CLR and adjusts the layer rate according to the new CLR.

Joining a layer involves additional multicast groups. SMCC achieves smoothness partly by using Binary Counting Layers (BCL), i.e., additional multicast groups, that are used to linearly increase the data rate. A receiver willing to join the next layer i must first join all BCL_j where $j <= i$ at times $t \cdot 2^j$. Right timing of BCL-joins result in linear increase in the data rate. The receiver performs the real join when the data rate has cumulated to the next layer rate.

While TFMCC may utilize the underlying TCP-model and adopt the TFRC-algorithm

with minor modifications to single-rate scenario, extending it to multi-rate scenario is not straight forward. In [KB03] this is identified:

> Loss rate measured by non-CLR receivers does not provide accurate information about the bottleneck bandwidth.

The authors remark this shortcoming in the results where receivers perform failed join-experiments due to inaccurate estimates of the rate.

SMCC poses a challenge to multicast routing protocol with the requirement of highly dynamic additional BCL layers. For a large session there is an additional BCL for each layer above the base layer. For each join experiment to a layer, all BCL below it must be joined. This leads to frequent changes in the multicast routing tree of the additional layers. The finer the layer granularity, the more frequently receivers change layers and the harder it is for multicast routers. We also find the protocol highly complex and hard to tune.

4 Applicability Analysis

We used TFRC as an instance of equation-based algorithm. The work suggested that the loss event rate sinks as a function of the sending rate leading to an overestimated rate. Motivated by the indication we studied the loss event rate computation in more detail.

We begin by discussing our simulation configuration to investigate open-loop TFRC behavior under varying conditions to show the cause and impact of the dependency on the sending rate. We then continue by studying the loss estimation algorithm or TFRC, used by TFMCC and SMCC. We identify the dependency on the sending rate in the algorithm. Using simulation we show results that support the analytical study. We then proceed with a recently introduced modified version of the loss estimation algorithm that works better with variable packet sizes and show that it does not remove the dependency on the sending rate.

Table 1: Simulation Factors and Levels.

Factor	Levels
Packet Loss Probability	0.1%, 0.5%, 1%, 3%, 5% and 10%
CBR Sending Rate	1, 2, 4, 8, 16, 32, 64 and 128 packets/RTT
Number of TCP Flows	1, 2, 4, 8, 16 and 32

4.1 Simulation Configuration

In this Section we describe our simulation configurations used to investigate the loss event rate and calculated rate behavior and dependency on the sending rate. In Section 3.1.4 we noted that the TCP-model of Equation 3 uses packet loss as congestion measure and assumes correlation among lost packets in a round. The basic time unit in the model is round-trip time. The sending rate is packets per round-trip time. We built our simulations around these basic units.

For the simulation configuration we identified four factors we consider important: sending rate; packet loss rate; degree of statistical multiplexing; and packet size. We discuss each of these in detail to provide the justification for the factors and the chosen levels of the factors. Table 1 provides a summary of the factors and levels used in the simulations.

We modified the NS-2 implementation of TFRC by breaking the closed-loop control into an open-loop control. We basically forced the sender to send at a constant bit rate (CBR) independent of the estimated level of congestion. However, we did let the end-systems to communicate the estimated round-trip time as in normal TFRC. Thus, the modification reminded much a unicast case where the sender may only choose some predefined rates and being unable to accurately follow the calculated rate. This reminds enough the multicast case in which only the frequent round-trip time estimates may not be carried out. As we shall show, the bias originates elsewhere and an accurate round-trip time estimate does not affect the bias.

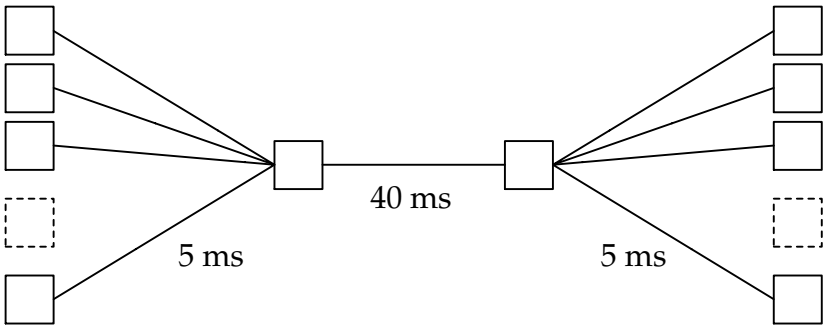


Figure 1: Dumbbell Topology.

4.1.1 General Setup

We limit our study to static scenarios. We assume that the bottleneck link is bandwidth limited and implements Drop-tail queuing policy. We also assume that the queue is measured in bytes instead of packets. That is, 10 packets of 100 bytes consumes as much buffer space as 1 packet of 1000 bytes. We chose Drop-tail queuing because the TCP-model assumes Drop-tail queues [PFTK98].

In the simulation setup we have one original closed-loop TFRC source (named TFRC), one modified open-loop TFRC source with CBR sending rate (named CBR) and a number of TCP/Sack sources (named TCP), depending on the experiment. All simulation experiments were repeated 15 times to collect enough traces to compute 95% confidence intervals for the means.

We configured a dumbbell topology (see Figure 1) with access link propagation delay of 5 ms and bottleneck link propagation delay of 40 ms. This sums up to a round-trip time of 100 ms, excluding queuing delays. Additional TCP sources were configured with access link propagation delay randomly taken from a window of 5-6 ms. We also made the TCP sources to start randomly in the first 20 seconds from the simulation start. Random propagation delays and start times were implemented in order to avoid phase-effects.

4.1.2 CBR Sending Rate

Our ultimate goal was to improve layer subscription decisions at the receiver of a layered multicast session where the sender emits data to layers at very slowly varying rates, or at constant bit rate (CBR). We model the behavior of a multicast source by configuring a TFRC source to transmit data at CBR rate independent of the congestion in the network. This effectively breaks the closed-loop nature of congestion control to an open-loop control and allows us to study the loss event rate computation and calculated rate under an open loop control.

We used exponentially distributed rates from 1 packet per round-trip time up to 128 packets per round-trip time (see Table 1). This range was enough to show the trend of the algorithms under study. With the propagation delay of 100 ms and at packet rate of 128 packets/RTT and Ethernet MTU of 1500 bytes the bitrate is around 1.5 Mbit/RTT. With packet size of 150 bytes we reach bitrate around 1.5 Mbit/s. Thus, the bitrates are reasonable for multimedia data. There is a wide range of conditions under which less than one packet per round-trip time is transmitted but we limit our study on scenarios where the packet rate is more than one packet per round-trip time. The packet sizes always include IP, UDP, TCP and application protocol headers.

4.1.3 Artificially Lossy Link

The process of forwarding packets of multiple sources to same output link in a router is called *multiplexing*. When this process is driven by demand of flows, the multiplexing is performed on a statistical basis and is called *statistical multiplexing*. In routers where large number of flows cross, the degree of statistical multiplexing is high and packet losses are more independent of each other. In packet-level simulator, such as NS-2, achieving high level of statistical multiplexing is slow since the simulator must keep state for each flow. Increasing the number of flows introduces more work to the simulator engine and significantly slows down the simulation process.

We compromised accurate statistical multiplexing and approximated it with artificially lossy link. With artificially lossy link we mean an over-provisioned link with large buffers and high link capacity and with a random packet dropper attached to the link. Different loss probabilities reflect different levels of congestion and let us study the algorithm behavior under a wide range of congestion levels that would be otherwise hard and slow to achieve in NS-2. We used Bernoulli loss model where all packet losses are independent of each other and varied the loss probability from 0.1% to 10%. The upper limit of 10% follows the recommendation of [BHH+00].

From the artificially lossy link configuration followed that round-trip time experienced by streams remained close to the round-trip propagation delay. Thus, the observed behavior were almost solely driven by the packet loss pattern. We note that the TCP-model in [PFTK98] assumes correlation of lost packets in a round. Under Bernoulli losses packets are dropped independent of each other and the assumption does not hold.

4.1.4 Shared Link

With shared link configuration we simulated low statistical multiplexing, i.e., scenarios where only a relatively small number of flows cross a bottleneck link. As the bottleneck router chooses packets of different flows to put in the output queue, it may only choose from the small set of flows available. As the flows "see each other" more frequently in the router, the flows are more likely to interfere with each other.

Shared link allowed us to study the protocol behavior under low degrees of statistical multiplexing when the sources compete on the available bandwidth. We varied the degree of statistical multiplexing by configuring N TCP flows, in addition to the TFRC/CBR and original TFRC flow. We varied N exponentially from 1 to 32.

We configured the bottleneck queue to approximately 1.5 times the bandwidth-delay product leading to a maximum queuing delay of 150 ms. We configured the bottleneck bandwidth such that the theoretical fair rate was 16 packets per round-trip time, with a reference packet size of 1000 bytes.

This configuration also had some drawbacks. Since the bottleneck bandwidth was configured to $(N + 2) \cdot 16$ packets per round-trip time, it effectively limited the number of reasonable scenarios for low values of N. For example, for a $N = 1$ the bottleneck bandwidth was 48 packets per round-trip time. Simulating CBR sending rate of 128 packets per round-trip time is above the link capacity and is of no value. Hence, we only simulated scenarios where the CBR sending rate was less than half of the bottleneck bandwidth. This can be seen in some 3-dimensional plots in the following sections where the missing points have been manually set to zero.

4.1.5 Packet Size

To achieve a throughput T, a source can send N_α packets of size s_α, or $N_\beta > N_\alpha$ packets of size $s_\beta < s_\alpha$, such that $T = N_\alpha s_\alpha \approx N_\beta s_\beta$. TCP usually sends packets adjusted to the path maximum transmission unit (MTU). Recall the discussion on TCP-compatibility in Section 3.1.5. We would like to achieve the same throughput as TCP on the same path regardless of the packet size. To study the relationship between packet size, computed loss rate and calculated rate we decided to run simulations with two packet sizes: 100 bytes and 1000 bytes. However, due to time and space constraints we decided to focus our effort on analyzing the results of simulations with packet size of 1000 bytes.

4.2 Equation-Based Rate Estimation

Although TCP has proven to scale way beyond the expectations and has kept the Internet functional, not all application domains can easily benefit from TCP con-

gestion control algorithm. The tightly coupled reliability is not always of greatest interest and the window based control is not always welcome. Especially application domains that require smooth rate changes, such as audio and video, are easier to control through rate change than congestion window manipulation.

TCP-Friendly Rate Control (TFRC) is a TCP-compatible congestion control algorithm for unicast flows [FHPW00b]. It attempts to provide smoother rate control suitable for multimedia applications, while still providing TCP-compatibility. It has got considerable attention in the past years and it has been intensively studied.

TFRC has been extended to single-rate multicast in TFMCC [WH01]. TFMCC has been extended from single-rate to multi-rate in SMCC [KB03]. Both of the protocol extensively rely on the functionality introduced by the original unicast TFRC. Especially, the loss estimator is left in original form.

It is crucial to understand how TFRC has been developed to operate and how it has been extended to multicast scenarios. For this reason we explain in detail the protocol internals before we proceed to the simulation results. Note that SMCC employs TFMCC in all layers. Modification of TFMCC apply SMCC, too.

4.2.1 Round-Trip Time Estimation

The self-clocking mechanism in TCP triggers the sending of packets. Each acknowledgment is treated as a clock tick and as a permission to send a packet. For a packet to be acknowledged a round-trip time elapses, which slows down the aggressiveness of a TCP flow. TCP throughput is inversely proportional to the round-trip time and it is important to have an accurate estimate of it in equation-based rate estimation.

In TFRC the receiver regularly feeds report messages back to the sender. The feedback messages carry the timestamp from the latest received packet S_i and the time elapsed between reception time and reporting time. The sender uses the

echoed timestamp to measure a round-trip time sample as in Equation 4

$$t_{RTTsample} = t_r^i - t_s^i - t_d^i \tag{4}$$

where t_s^i is the time at which packet S_i was sent and t_r^i is the time at which the feedback message carrying the echoed timestamp was received at the sender and t_d^i is the delay between reception time and feedback sending time at the receiver.

An Exponential Moving Weighted Average (EMWA) is used to smooth the round-trip time estimate as in Equation 5

$$t_{RTT} = q \cdot t_{RTT} + (1 - q) \cdot t_{RTTsample} \tag{5}$$

where q is a filtering constant. The recommended value for q is 0.9 [HFPW03]. The larger q the less reactive to new round-trip time samples the estimation will be. Using EMWA is practical because it requires no history of previous samples of round-trip time.

Each round-trip time sample will be used to compute a new estimate for retransmission timeout (t_{RTO}) which is required by the TCP-model. Retransmission timeout value is computed simply as a multiple of round-trip time sample as in Equation 6.

$$t_{RTO} = 4 \cdot t_{RTTsample} \tag{6}$$

In multicast congestion control the closed-loop approach is not feasible because of feedback implosion and scalability constraints. TFMCC adopts the round-trip time estimation as it is specified for TFRC, but applies it only to the group member that has the lowest estimate of the calculated rate. TFMCC sender chooses such a sender as the *Current Limiting Receiver* (CLR) and forms a closed loop control with it. The session bandwidth is then adjusted according to the CLR. Other members in the group send feedback to the sender only rarely. This leads to an inaccurate round-trip time estimate.

4.2.2 Loss Fraction

Congestion in the Internet is signaled to sources by routers. TCP interprets packet losses (or marked packets) as congestion signals and reacts to them by halving the congestion window. However, equation-based congestion control directly adjusts the sending rate and does not have a concept of congestion window. The TCP-model expressed in Equation 3 takes the window-halving behavior into account. Given a steady packet loss rate, it computes a long-term TCP throughput. Thus, it is crucial to accurately estimate the loss rate to feed in the TCP-equation.

Loss fraction, in general, is the fraction of packets lost over a time frame and it indicates the amount of congestion in the network. The simplest way to compute loss fraction is to compute the number of lost packets of the number of transmitted packets. This is given in Equation 7 and is is called *packet loss rate*.

$$p = \frac{N_{lost}}{N_{transmitted}} \tag{7}$$

We use the term loss fraction to refer to packet losses in general. When we refer to TCP, we use term packet loss rate and when we refer to TFRC, we use the term loss event rate. In Section 5 we modify the TFRC loss event rate algorithm but we still call it loss event rate, although it is a slightly inaccurate term for it.

4.2.3 Loss Event

TCP reacts to, one or more, lost packets in a round by halving the congestion window. That is, TCP reacts to any number of losses in a round time only once. TFRC imitates this by aggregating packet losses to *loss-events*. TFRC effectively ignores additional packet losses in a round-trip time after the initial packet loss. A new loss event is triggered if at least one round-trip time has elapsed since the begin of the last loss event. We adopt the term *Loss Insensitive Period* (LIP) introduced by [WBB04] to refer to the round-trip time following the initial packet loss. Figure 2 illustrates the LIP.

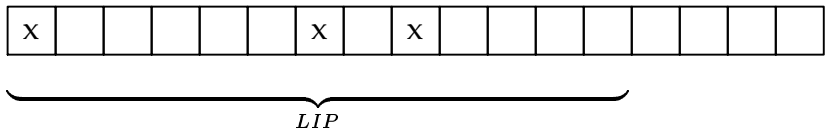


Figure 2: Loss insensitive period (LIP). Lost packets are marked with x.

It is important to notice that no matter how many real packets get lost in a loss event, the impact is always constant, i.e., as if only one packet had been lost. Thus, the nth loss event Ψ_n is always 1 for all n.

If only one packet is lost in every loss event, then $\sum \Psi_i$ equals to the number of lost packets N_{lost}. However, if more than one packet is lost in some loss event, then the number of loss events is smaller than the number of truly lost packets. That is, measuring losses as loss events may result in a weaker signal of congestion than using true packet loss rate.

Assuming a Bernoulli loss model with packet loss probability p, a flow sending N packets in a round-trip time has the probability of

$$p_{packet} = 1 - (1 - p)^N \tag{8}$$

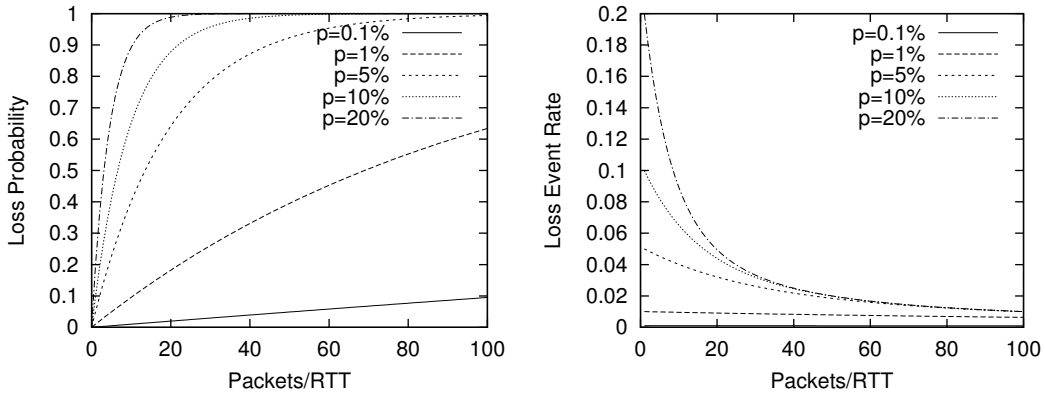
to experience at least one packets loss in a round-trip time [FHPW00a]. Loss event rate, computed as loss events per packets sent, is given in Equation 9.

$$p_{event} = \frac{1 - (1 - p)^N}{N} \tag{9}$$

Floyd et al. note in [FHPW00a] that for a fixed loss probability, the faster the sender transmits, the lower the loss event fraction. Figure 3 shows that the loss event rate indeed sinks as the number of packets per round-trip time increases.

4.2.4 Loss Interval

Loss interval is the number of transmitted packets between two consecutive loss events. All packets, whether successfully received or lost, are part of exactly one



(a) Equation 8. *(b) Equation 9.*

Figure 3: Packet loss probabilities.

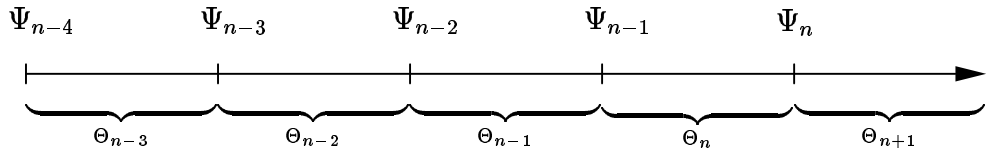


Figure 4: Loss events and loss intervals.

loss interval. The lost packet that triggers a loss event may be counted to the loss interval preceding the loss event, or to the loss interval that follows the loss event, but not to both. Also the packets during a LIP are counted to a loss interval.

Assume that packet S_a triggers a loss event Ψ_{n-1} and packet S_b triggers the next loss event Ψ_n. Then, loss interval Θ_n is $b - a$. The sum of all loss intervals is the number of transmitted packets. The sum of loss intervals is analogous to the $N_{transmitted}$ of the Equation 7. Loss intervals, including the current non-terminated loss interval, are illustrated in Figure 4.

By definition, a loss interval is always initiated and terminated by loss events. However, the packets transmitted after the most recent loss event is also considered as a loss interval, too, although it is not terminated by a loss event. It will continuously grow, until terminated by a loss event.

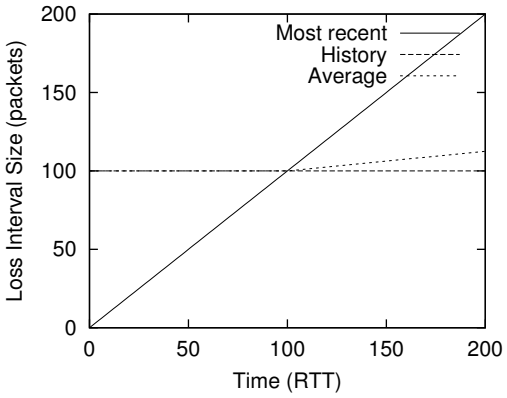


Figure 5: Development of average loss interval.

4.2.5 Loss Event Rate

Loss event rate is basically the frequency of loss events. The more frequently loss events are triggered, the more there is congestion in the network. In [FHPW00b] several loss event rate algorithms were evaluated. The authors concluded that Weighted Average Loss Interval (WALI) provided best results. WALI was also chosen for TFMCC and SMCC, as well as included in TFRC specification [HFPW03].

In WALI k recent loss intervals are used in computation. The most recent, non terminated, loss interval is referred to as Θ_0. The first terminated loss interval is Θ_1 and the kth recent loss interval is Θ_k. The loss intervals are weighted using an Exponential Weighted Moving Average (EWMA) of k weight factors such that the recent intervals weight more than old intervals.

$$\Theta_{avg} = \frac{max\left(\sum_{i=0}^{k-1} w_i\Theta_i,\ \sum_{i=1}^{k} w_{i-1}\Theta_i\right)}{\sum_{i=0}^{k-1} w_i} \tag{10}$$

The output of the Equation 10 is illustrated in Figure 5. The Figure shows the intersection of the history average loss interval and the most recent non-terminated loss interval. As the most recent loss interval grows enough, it starts affecting the average loss interval. This way TFRC smoothly accounts the growing non-terminated loss interval.

By manipulating the history size k and list of weights TFRC can be made more or less aggressive to dynamic conditions in the network. The authors of TFRC

recommend using a value of k not significantly greater than 8 and using weights (1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2). The authors of TFMCC note that values of k around 8 to 32 appear to be a good compromise and provide, as example, weights (5, 5, 5, 5, 4, 3, 2, 1), which is equivalent to TFRC weights.

The loss event rate, which is the measure of congestion, is then computed as inverse of the average loss interval

$$p_{event} = \frac{1}{\Theta_{avg}} \tag{11}$$

where Θ_{avg} is from Equation 10.

4.3 Simulation Results

The loss event rate computation algorithm of TFRC and TFMCC depends on the length of the average loss interval. The averaged loss interval depends on the length of individual loss intervals. An individual loss interval depends on when two consecutive loss events are triggered. Packet losses and loss events depend on the loss pattern which depend on, among others, source behavior and bottleneck link. In our simulations, the loss pattern depends on the artificially lossy link configuration, or on the number of competing TCP sources.

We analyze the results from three perspectives. We first study the behavior as a function of artificial congestion over the simulated artificially lossy link. Then we analyze the results as a function of number of competing TCP flows over a shared link. And third, we analyze the behavior as function of the CBR sending rate. Due to space constraints we present only a fraction of the plots[4]. We have picked plots that illustrate the observations made from the whole set of plots.

4.3.1 Artificial Congestion

Figure 6 shows an overview of the loss event rate and calculated rate as a function of CBR sending rate and artificial packet loss probability. The Figure shows

[4]We generated about 150 plots from several gigabytes of simulation data.

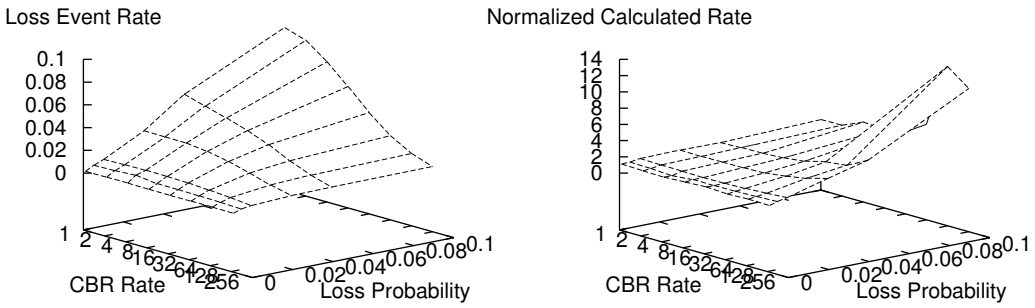


Figure 6: Overview of loss event rate and calculated rate under artificial congestion.

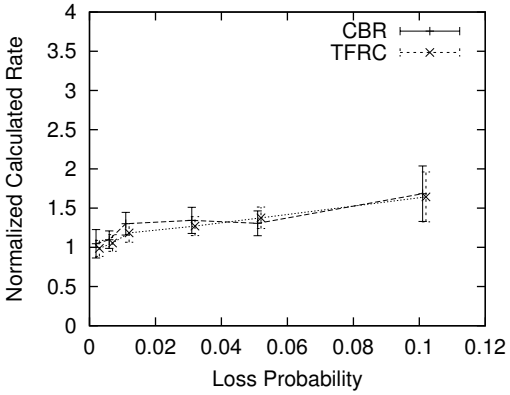


Figure 7: CBR rate of 1 packet/RTT over artificially congested link.

that the estimated loss event rate grows linearly with the packet loss probability only at very low sending rates. As the sending rate and packet loss probability increase, the estimated loss event rate starts to sink down.

At low packet rates and low packet loss rates two consecutive packet losses have a low probability to occur close to each other. Thus, two consecutive loss event will have a large span and loss intervals will be large leading to a large average loss interval. In such circumstances the loss event rate almost equals true packet loss rate. Figure 7 shows that loss event rate and calculated rate as a function of packet loss probability when the CBR sending rate is only one packet/RTT. All plots show mean values with 95% confidence intervals.

At high packet rates and modest packet loss rates it is possible to experience packet losses within a small time frame. Recall that a loss event follows a LIP

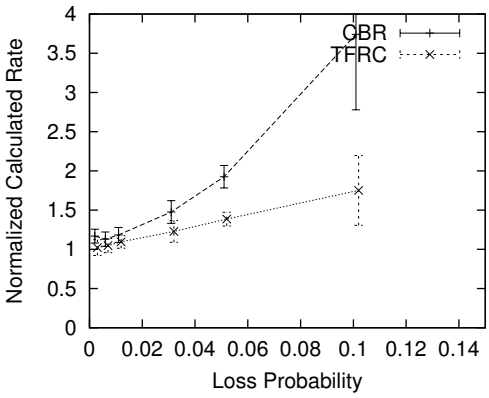


Figure 8: CBR rate of 16 packet/RTT over artificially congested link.

during which all packet losses are ignored. However, they are accounted in the next loss interval. When the packet rate and packet loss rate cross, such that packet losses are experienced frequently, then new loss events are triggered soon after the LIP.

In such a scenario, as the loss interval averaging is bound to a k recent loss intervals, frequent loss events quickly drive the loss interval history down to cover k round-trip times only. Thus, the average loss interval becomes dependent on the number of packets per round-trip time, i.e., the number of packets sent in a LIP. The dependency can be seen in Figure 8. We return to the issue of LIP dependency in Section 4.3.4.

4.3.2 Congestion due to Competition

Figure 9 provides an overview of loss event rate and calculated rate under shared link environment as a function of number of competing TCP sources and CBR sending rates. The underestimation of loss event rate at high CBR rates is smaller compared to the artificially generated congestion. This is due to back-off of competing flows. Thus, the congestion never became as bad as was artificially generated in the previous simulations.

When the congestion occurs in a router with low degree of statistical multiplexing, the CBR flow experiences proportionally higher loss event rate at low send-

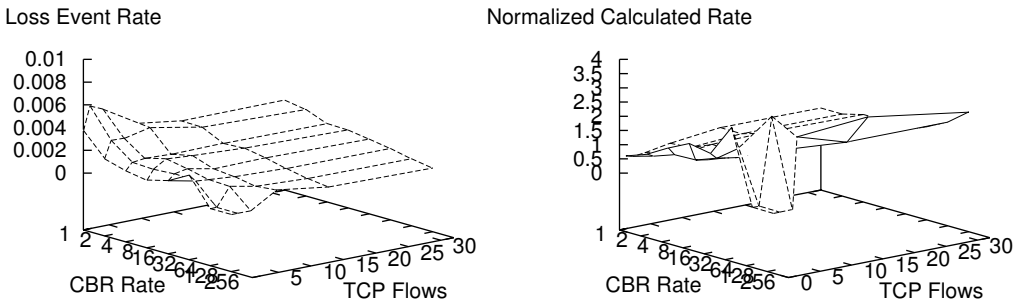


Figure 9: Overview of loss event rate and calculated rate under competition. NOTE: The values at high CBR rate with few TCP flows are manually set to zero. See Section 4.1.4 for details.

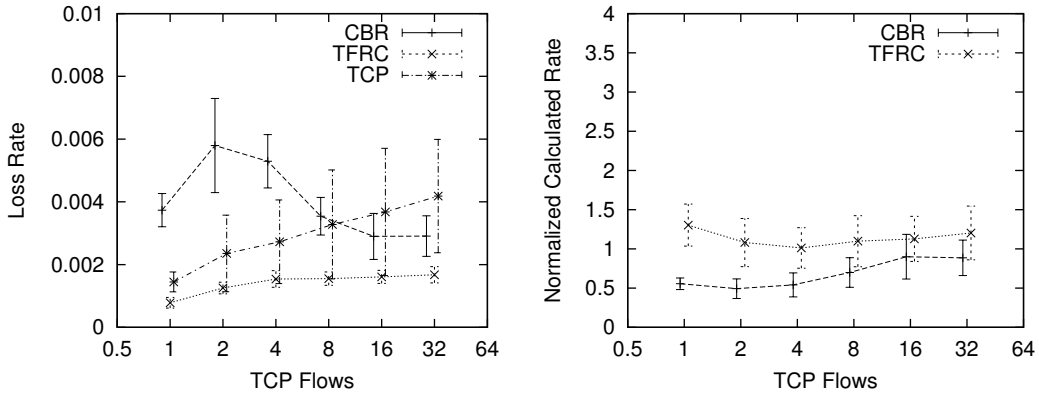


Figure 10: Shared link with CBR sending rate of 1 packet/RTT.

ing rates than a comparable TFRC flow or TCP flow. The competing flows grab the available bandwidth and populate the Drop-tail queue in the bottleneck router. As the CBR flow only rarely (compared to competing flows) sends a packet, a single drop easily leads to an overestimate of the real congestion. Under low degree of statistical multiplexing it appears to be quite hard to do accurate estimates.

Figure 10 shows how the loss event rate varies as a function of the number of competing TCP flows when the CBR rate is 1 packet/RTT. For very low degrees of statistical multiplexing the CBR flow makes overestimates of the loss event rate and calculates a low rate. As the multiplexing level increases, the calculated rate and TFRC sending rate start to converge to the TCP throughput.

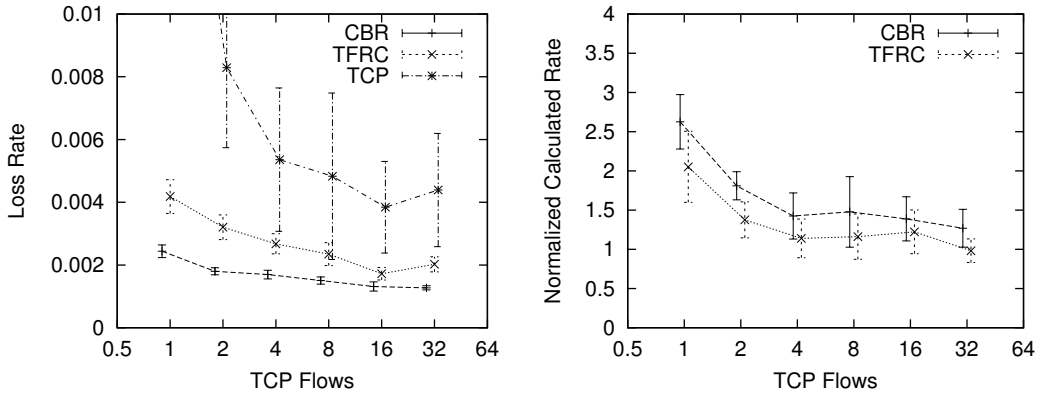


Figure 11: Shared link with CBR sending rate of 32 packet/RTT.

Figure 11 shows the loss event rate and calculated rate as a function of the number of competing TCP flows competing when the CBR sending rate is 32 packets/RTT. Independent of the number of competing TCP sources, the loss event rate remains too low and the calculated rate too high. In fact, as the number of competing sources increase, so does the congestion, and then the estimated loss event rate decreases. This supports the behavior observed under artificially lossy link, i.e., as the packet loss probability increases, the estimated loss event rate decreases. Note that in Figure 11 the TCP packet loss rate at left is above the Figure border.

4.3.3 Sending Rate

We have already shown how the loss event rate and calculated rate depend on the CBR sending rate over a wide range of different types of congested links. We also remarked that low degree of multiplexing appear to be troublesome to rate estimation. We now analyze the results from the perspective of different sending rates, i.e., how does the estimated loss event rate change as a function of the CBR sending rate.

The configuration of artificially lossy link allows us to study the loss estimator under well controlled congestion scenarios. For low CBR sending rates and low packet loss rates, the estimator works like normal TFRC and gives results compa-

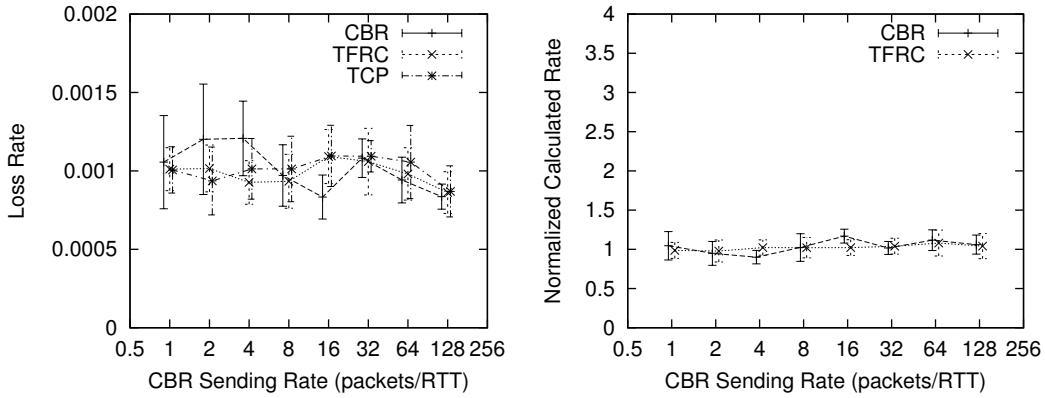


Figure 12: Artificially lossy link with 0.1% packet loss probability.

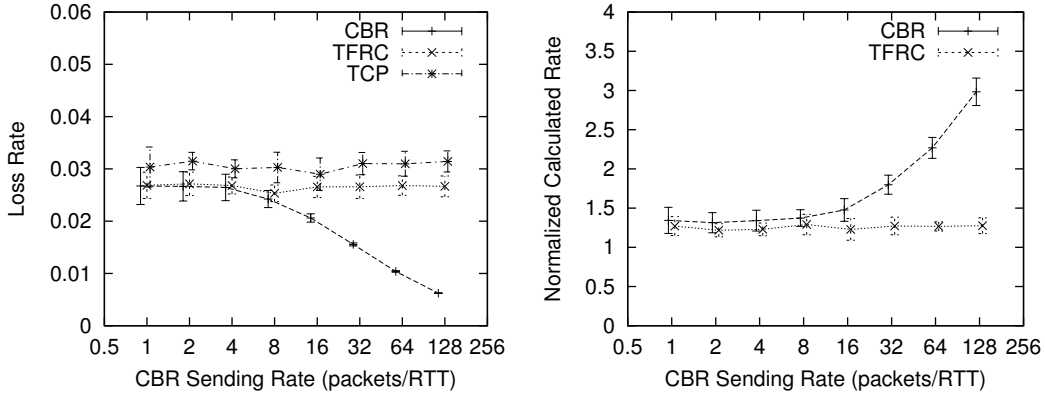


Figure 13: Artificially lossy link with 3% packet loss probability.

rable to TFRC and TCP. That it, the CBR flow follows the TFRC and TCP flows. This can be seen in Figure 12.

When the link is configured to 3% packet loss probability, the loss event estimator becomes heavily dependent on the CBR sending rate, i.e., on the number of packets/RTT. The effect can be seen in Figure 13. The estimated loss event rate sinks rapidly leading to a too high calculated rate. This supports the results presented previously.

Under shared link configuration the CBR flow shows peculiar behavior under very low degrees of statistical multiplexing. Figure 14 shows the estimated loss event rate and calculated rate as a function of sending rate when competing with 1 TCP flow. At very low sending rates the estimated loss event rate is too high. At high sending rates it is too low.

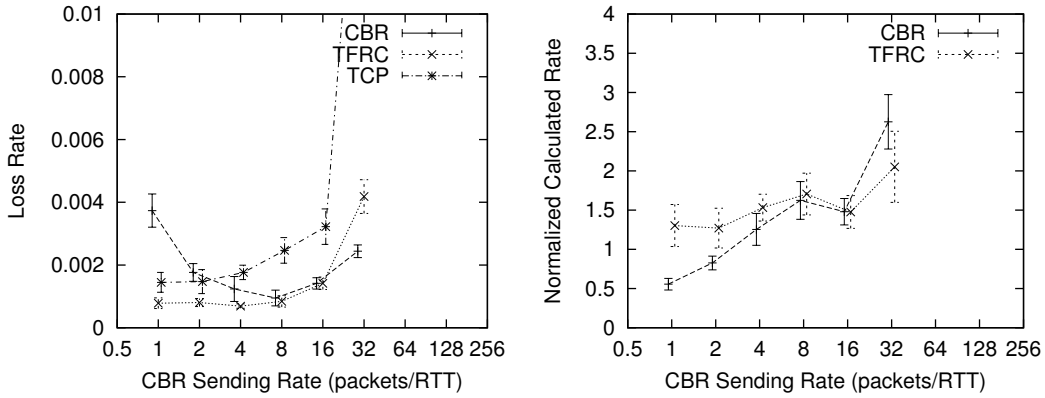


Figure 14: Shared link with 1 TCP flow.

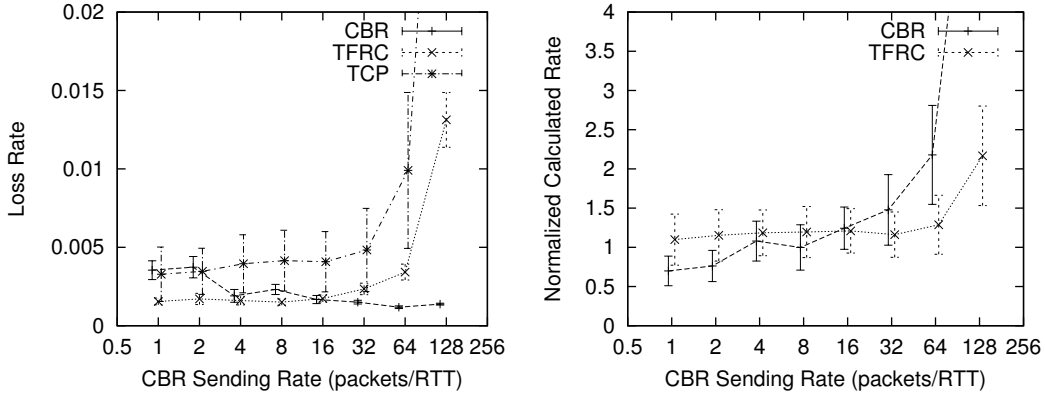


Figure 15: Shared link with 8 TCP flows.

The effect of growing the degree of statistical multiplexing by increasing the number of competing TCP sources is shown in Figure 15. The overestimation of loss event rate is still visible at very low sending rates. However, it quickly converges with TFRC as the sending rate increases, and, eventually provides a significantly too low estimate leading to overshooting calculated rate.

Recall from Section 4.1.4 that the theoretical fair rate was 16 packets per round-trip time. Interestingly, CBR and TFRC converge at the theoretical fair rate of 16 packets per round-trip time. Thus, under relatively low degrees of statistical multiplexing, the CBR appears to provide good estimates only when it happens to be sending at the rate that is close to the flow's fair share of the bottleneck link.

4.3.4 Virtual Packets

During our work we became aware of a contribution of Widmer et al. that changed the TFRC loss event part of the algorithm to improve the loss event computation in presence of variable size packets [WBB04]. For a bitrate R the packet rate of small packets is larger than big packets. Assume a source sends N packets of size s_α in a round-trip time. To achieve the same bitrate using packet size $s_\beta < s_\alpha$, more packets must be sent. The relationship between two different packet sizes at same bitrate is

$$R = Ns_\beta = \alpha N s_\alpha$$

where $\alpha = s_\beta / s_\alpha$.

The number of small packets that fit in a LIP is larger than the number of big packets for the same bitrate. It follows that the loss interval for small packets is larger than for big packets and it follows that the loss event rate is smaller. Thus, the measure of congestion depends on the packet size.

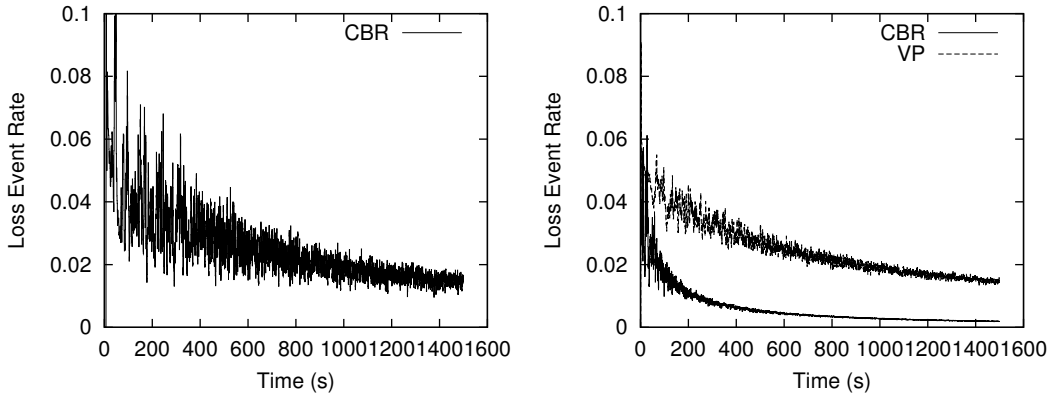
Widmer et al. derive the expected value for an individual loss interval

$$E(\Theta) = N - 1 + \frac{1}{p} \tag{12}$$

where Θ is the loss interval, $N - 1$ is the number of packets sent in a LIP after the initial packet loss and p is the packet loss probability. The Equation 12 shows that the loss interval becomes dependent on the number of packets N sent in a round-trip time.

They propose three different algorithms to handle variable size packets and show that the method of virtual packets (VP) provides best results. In the VP method real packets are mapped to entities called virtual packets and all computation is done using the virtual packets. The method achieves equal loss event rates for flows using different packet sizes.

The Figure 16 shows that VP flow, with real packet size of 100 bytes and virtual packet size of 1000 bytes, achieves an estimate similar to a CBR flow using packet size of 1000 bytes. This supports the analytical study of the VP method. Since a



(a) Packet Size 1000 Bytes. *(b) Packet Size 100 Bytes.*

Figure 16: Loss event rate as sending rate was increased under artificial congestion of 5% packet loss probability. In (b) above VP and below CBR.

VP flow is comparable to a CBR flow with matching virtual packet size and real packet size, respectively, we decided not to run the large simulation set for the VP method. Instead, we base our conclusions on the analytical study and the results of normal CBR.

However, the basic concept of loss event rate suffers from the fundamental problem of LIP and loss interval computation in case of an open-loop sender. Replacing N in Equation 12 with the number of virtual packets $N_{virtual}$ (of size $s_{virtual}$), it can seen that the loss event rate computation still depends on the sending rate. Then, it depends on the sending rate of virtual packets being equivalent to an open-loop TFRC sending packets of size $s_{virtual}$. TFRC with LIP is always bound to the N, be it real or virtual packets. This is also visible in Figure 16 where the loss event rate of the VP flow sinks in time, i.e., as the number of virtual packets per round-trip time increases.

4.4 Summary

In this Section our primary interest was to study the TFRC rate estimation algorithm applicability to open-loop control scenario. For the study, we modified the

TFRC algorithm by breaking the closed loop control into a half open-loop control. We let the receiver send feedback but forced the sender to send at configured CBR sending rate. Using the modified TFRC simulator implementation we conducted a set of simulations to study the calculated rate dependency on the CBR sending rate.

We developed a simulation configuration that let us study the behavior under wide range of conditions. We exponentially varied the CBR sending rates from 1 to 128 packets per round-trip time to investigate the loss event rate and calculated rate dependency on the sending rate. The simulated scenarios included artificial congestion that attempted to capture environments with very high degree of statistical multiplexing. For environments with low degree of multiplexing, we used TCP/Sack flows that competed over a shared link.

We also described in detail the TFRC loss event rate algorithm. TFRC measures congestion as loss event rate. A packet loss triggers a loss event. A loss event follows a Loss Insensitive Period (LIP) during which all packet losses are ignored. The number of packets transmitted between consecutive loss events is called loss interval. The loss event rate is the inverse of an average loss interval over a history of k recent loss intervals.

We showed that as the packet loss rate and sending rate increase, the loss estimation algorithm becomes dependent of the CBR sending rate. The higher sending rate or packet loss rate, the lower loss event rate the algorithm computes leading to overshoots in calculated rate. We also showed that the recent contribution of Widmer et al. working on variable packet sizes does not solve the biased estimation. We also identified the origin of the bias to the Loss Insensitive Period which ignores packet losses after the initial packet loss triggering the loss event.

5 Modification to Loss Fraction Computation

In this Section we present a modification to the loss event rate algorithm introduced by TFRC, and used by TFMCC and SMCC. Instead of the Loss Insensitive Period (LIP), where packet losses following an initial packet loss are ignored, we aggregate them to measure an impact of the loss event. Loss event rate with impacts imitates packet loss rate. We judge this partly by the assumptions of the underlying TCP-model and, partly by the requirement to decouple sending rate from loss estimation. We start by discussing the modification and then proceed with the simulation results.

5.1 Loss Aggregation Period

To acquit of the biased loss event rate computation of TFRC we modify it. Our motivation to do so is based on (1) the requirement to decouple the loss fraction computation from the sending rate, and (2) the different assumptions present in open-loop scenario and the underlying TCP model. The modification is isolated to the LIP part of the algorithm and we call it *Loss Aggregation Period* (LAP). It effectively aggregates packet losses during a round-trip time after an initial loss by counting the number of lost packets. This gives a measure we call *loss impact*.

5.1.1 Assumptions of the TCP Model

Packet loss fraction is effectively a measure of congestion signaled by routers on an end-to-end path. The TCP-model presented in [PFTK98] uses packet loss rate as measure of congestion and models the long-term TCP throughput on a path with that level of congestion.

The TCP-model divides time in rounds. A round equals to the current round-trip time. In the beginning of each round, a full window of back-to-back packets is sent. Packet losses in different rounds are assumed to be independent of each

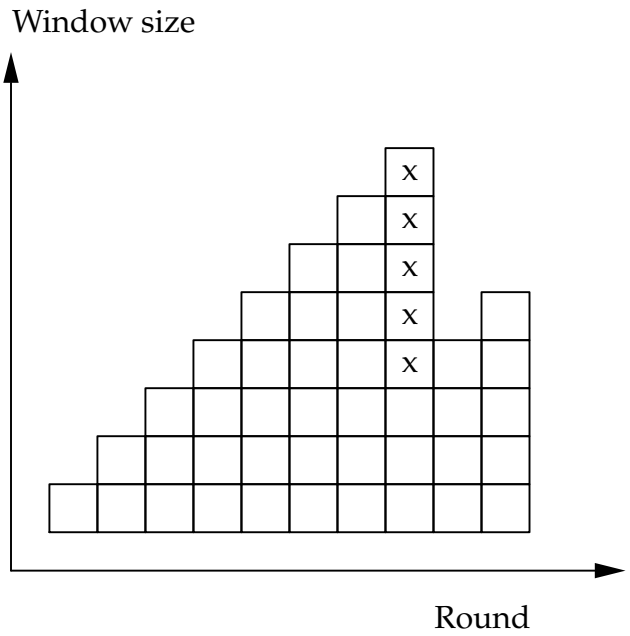


Figure 17: TCP-model assumption of packet loss correlation.

other but a packet loss in a round is considered to cause loss of all of the *remaining* packets in the round, which we call *virtually lost packets*, and is illustrated in Figure 17. This assumption originates from Drop-tail queuing policy where all back-to-back packets following an initial packet loss are likely to be dropped, too. As congestion measure the model uses the sum of truly and virtually lost packets.

In fact, a packet loss followed by virtual packet losses in a round constitutes a loss event. However, in the model, the impact of a loss event depends on the number of transmitted packets per round-trip time. The more packets are transmitted, the higher the impact is. TFRC simply ignores all losses in LIP and effectively assumes a constant impact of one lost packet. However, blindly following the model and counting the rest of the packets in a round as lost would make the algorithm directly dependent on the sending rate. And that's precisely what we want to acquit of.

Floyd et al. note in [FHPW00a] the following:

> The version of the TCP response function is based in some respect to the loss event rate, and in other respects on the packet loss rate. [...] Ideally, this response function would be replaced with a TCP response function based on a model of Sack TCP and on loss event rates than on packet drop rates.

The concept of loss event rate is judged with the way TCP reacts to packet loss, i.e., by halving the congestion window once per round-trip time, regardless how many packets really get lost in the round-trip time. However, the TCP-model used by TFRC assumes packet loss rate and explicitly models the window halving behavior of TCP. The window halving is, kind of, accounted twice: first in TFRC loss event rate computation, and then in the TCP-model. Also, in rate controlled flows packets are sent in certain frequency. Thus, there are no back-to-back packets, as assumed by the underlying model.

Despite the differences in implementation and assumptions, our results confirm that TFRC shows good behavior under closed-loop control. However, it leaves room for optimization when applied to an open-loop scenario.

5.1.2 Origin of the Dependency

Equation-based rate estimation uses a rate equation to compute a sending rate. In case of TFRC the rate estimation is based on a model of long-term TCP throughput [PFTK98]. The formula expects parameters to describe the circumstances under which an estimated TCP source operates. One of the parameters is packet loss rate that indicates the amount of congestion in the network. The lower the packet loss rate, the higher the calculated rate.

As we broke the closed-loop control of TFRC to an open-loop control we observed a bias in the loss fraction computation. The more packets were sent, the less congestion was estimated. As a result, the TCP formula was fed in a lower estimate of network congestion resulting in a higher estimate of sending rate.

The biased loss estimation originates from the TFRC loss event rate algorithm in general, and from LIP in particular. In LIP, all packet losses following an initial packet loss are ignored for a time frame that equals the round-trip time. All packets in a LIP are counted for the next loss interval and loss intervals are used to estimate loss event rate. If a source send N packets per round-trip time, then a loss interval between two consecutive loss events is always at least N. Since N directly depends on the packet sending rate, the loss estimator becomes dependent on the sending rate.

5.1.3 The Modification

We call our modified loss event rate algorithm *Loss Aggregation Period* (LAP). We call the original algorithm *Loss Insensitive Period* (LIP). The difference between LIP and LAP is that whilst LIP *ignores* packets, LAP *aggregates* them to constitute a *loss impact*. That is, after the initial packet loss that triggers a loss event, we count all packet losses during the following round-trip time. The number of lost packets during the round-trip time is the loss event impact. We also define that the nth loss interval Θ_n is terminated by nth loss event of impact Ψ_n, except for the current non-terminated loss interval.

Let

$$lost : \mathbb{N} \rightarrow \{0, 1\} \mid lost(i) = \begin{cases} 0 & S_i \text{ was transmitted successfully} \\ 1 & S_i \text{ was lost} \end{cases}$$

Assume that a source sends N packets per round-trip time and packet S_a is the first packet of loss interval Θ_n and that packet S_b triggers the nth loss event. Then, the nth loss interval is

$$\Theta_n = b + N - a \tag{13}$$

and the loss impact of nth loss event is

$$\Psi_n = \sum_{i=b}^{b+N-1} lost(i) \tag{14}$$

If only one packet is lost in nth loss event, then $\Psi_n = 1$, and if 7 packets are lost in mth loss event, then $\Psi_m = 7$. The mean[5] loss impact of loss events over a history of k recent loss intervals is

$$\bar{\Psi} = \frac{1}{k} \sum_{i=0}^{k-1} \Psi_{n-i} \tag{15}$$

Average loss impact over a history of k loss intervals with weighting is

$$\hat{\Psi} = \frac{\sum_{i=0}^{k-1} \Psi_{n-i} w_i}{\sum_{i=0}^{k-1} w_i} \tag{16}$$

where weights w_i are the same as used in the original algorithm (see Section 4.2.5 for details of weights).

To compute the loss event rate with loss impact we modify the loss event rate computation. The modified computation is presented in Equation 17.

$$p_{event} = \frac{\bar{\Psi}}{\Theta_{avg}} \tag{17}$$

Since $\bar{\Psi}$ and $\hat{\Psi}$ are analogous to N_{lost} and Θ_{avg} is analogous to $N_{transmitted}$ of Equation 7, we effectively imitate the same equation, i.e., packet loss rate, averaged over a history of k recent loss intervals.

5.2 Simulation Results

We used the same simulation configuration as in the applicability analysis (see Section 4.3), now including the modified loss event rate algorithm. We also use the same structure in presenting the results and show the same set of figures to allow easy comparison. In the plots, the new algorithm is referred to as LAP and the old algorithm is referred to as CBR, as in previous plots. The CBR flow implements LIP. The plots show 95% confidence intervals collected from 15 runs.

[5]We modified the default TFRC implementation in NS-2 for the simulation. TFRC weights and discounts loss intervals but our modification used mean impact. The results nevertheless show the loss estimator independence of the sending rate. We later implemented LAP with weighting and use it in our multicast implementation.

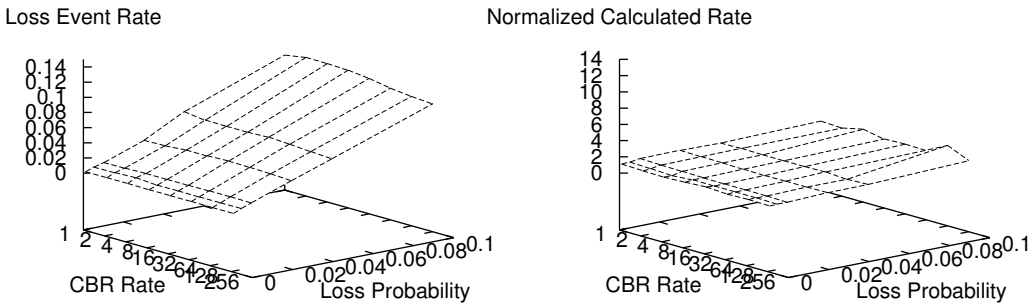


Figure 18: Overview of loss event rate and calculated rate under artificial congestion.

5.2.1 Artificial Congestion

Figure 18 shows an overview of loss fraction and normalized calculated rate as a function of CBR sending rate and artificial packet loss probability. The Figure shows how the estimated loss event rate grows linearly as function of the packet loss probability independent of the CBR sending rate. Also, the normalized calculated rate shows no dramatic unfairness over the whole region of parameter settings.

At low CBR sending rates, LAP shows almost equal behavior to CBR and TFRC. In Figure 19, LAP closely follows CBR and TFRC at all packet loss probabilities when the CBR sending rate is only 1 packet per round-trip time. This is obvious since LAP operates on multiple packets per round-trip time. When the packet rate is only one packet per round-trip time, the behavior is equal to LIP.

At higher CBR sending rates, LAP plays a role because the number of packet per round-trip time is greater than one. This can be seen in Figure 20. The CBR loss event rate badly underestimates the congestion causing the sending rate to rocket up, while LAP follows the closed-loop TFRC. When the packet loss probability increases and more than one packet per round-trip time is sent, there may be more than one lost packet in a loss event. The fact that LAP measures the loss impact allows it to estimate an accurate measure of the congestion, also in extreme conditions.

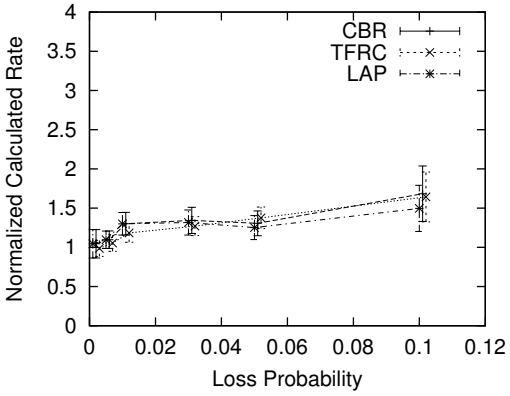


Figure 19: CBR rate of 1 packet/RTT over artificially congested link.

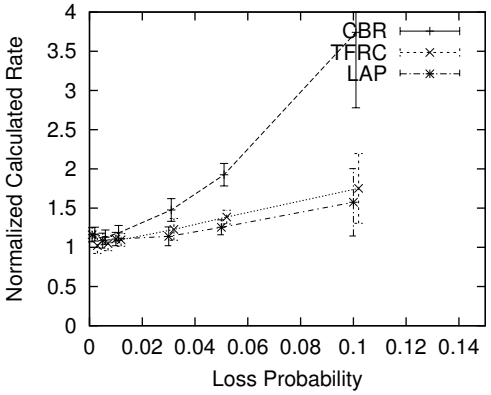


Figure 20: CBR rate of 16 packet/RTT over artificially congested link.

5.2.2 Congestion due to Competition

Figure 21 shows an overview of loss fraction and normalized calculated rate as a function of CBR sending rate and number of competing TCP sources. The Figure shows that at higher degrees of statistical multiplexing the estimated loss fraction and calculated rate are fairly good. They also show overestimation of loss fraction and low calculated rate at low CBR rates and very low degree of statistical multiplexing.

At low packet rates LAP loss event rate and calculated rate follow the CBR like under artificial congestion. The reason is the same as under artificial congestion: LAP operates with multiple packets per round-trip time. LAP plays no role at rates lower than, or equal to, one packet per round-trip time. This is visible in

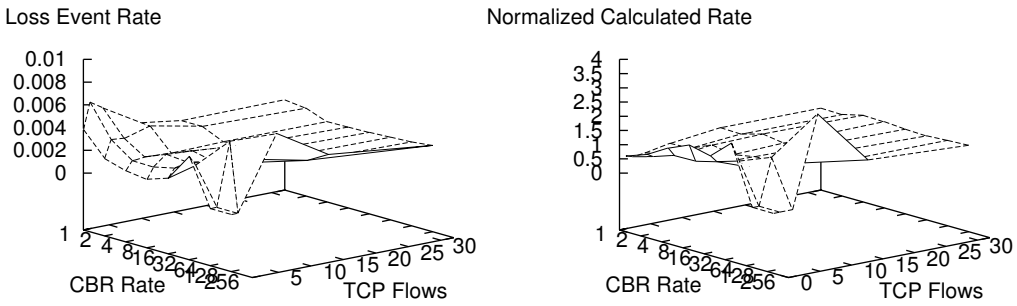


Figure 21: Overview of loss event rate and calculated rate under competition. NOTE: The values at high CBR rate with few TCP flows are manually set to zero. See Section 4.1.4 for details.

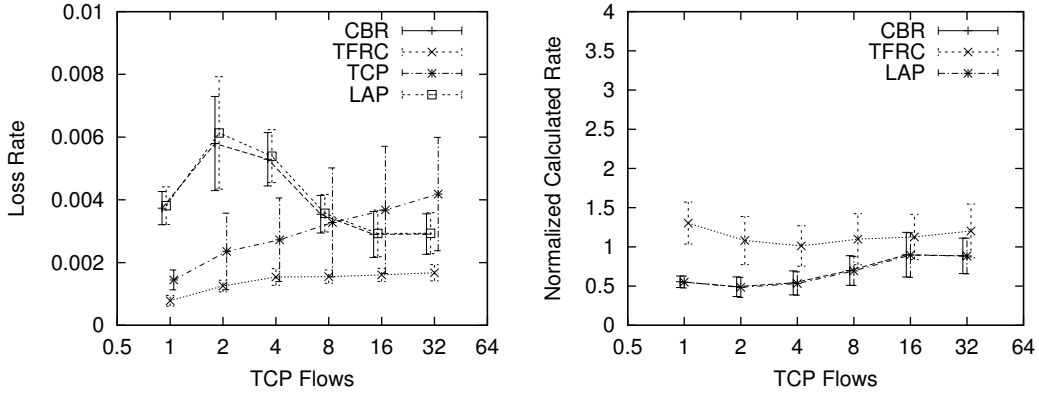


Figure 22: Shared link with CBR sending rate of 1 packet/RTT.

Figure 22.

It is worth emphasizing that LAP suffers from the same troubles as LIP under very low degrees of statistical multiplexing. At a low sending rate there are only rarely possibilities to get a congestion signal and in presence of Drop-tail queues the congestion signal is not equal to all flows, rather it depends on flow characteristics. Thus, the detected signal depends on the flow itself, or on competing flows. We return to the challenge of measuring a reliable congestion signal under very low degrees of statistical multiplexing in Section 5.2.4.

At higher CBR sending rates, LAP plays a role because the number of packet per round-trip time is greater than one. This can be seen in Figure 23. The CBR loss event rate again underestimates the congestion and calculates an unfair rate.

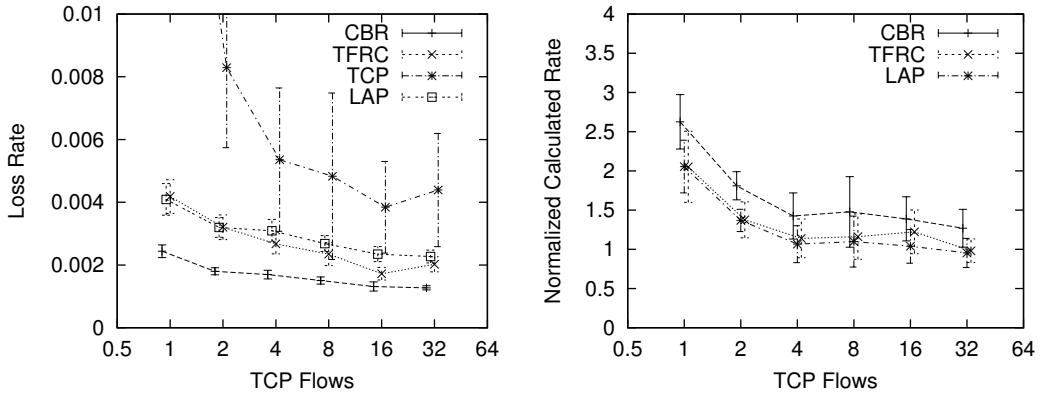


Figure 23: Shared link with CBR sending rate of 32 packet/RTT.

However, LAP accurately follows the closed-loop TFRC.

The calculated unfairness is partly due to our configuration where the bottleneck bandwidth is throttled to have a theoretical fair rate of 16 packets per round-trip time (see Section 4.1.4). In Figure 23 the CBR sending rate is 32 packets per round-trip time, i.e., twice the theoretical fair rate. With only a few competing TCP sources, the CBR (or LAP) takes a relatively large portion of the bandwidth and beats the TCP down.

5.2.3 Sending Rate

We have already shown that the LAP accurately measures congestion over a variety of congested links. We now study the results from the CBR sending rate perspective and show how LAP performs over a varying sending rates.

Figure 24 shows that LAP, CBR, TFRC and TCP all operate almost equally over a wide range of sending rates under 0.1% Bernoulli loss model. All of the equation-based algorithms give quite accurate estimates of the congestion leading to a fair calculated rate. The packet loss probability is so low and the packet rate low enough that a packet loss is experienced only rarely.

We next adjusted the packet loss probability to 3%. At that packet loss probability, on the average, every 33th packet is lost. Theoretically, the effects of CBR ignoring packets during LIP should be visible starting from rates above 33 packets per

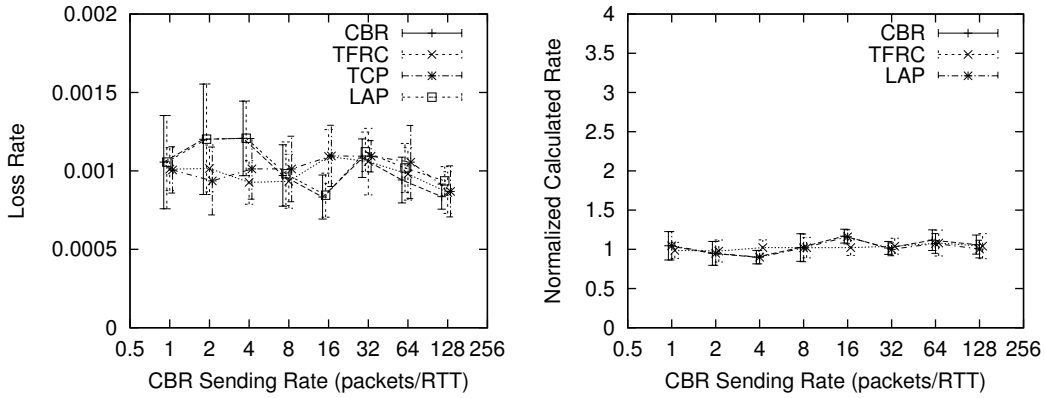


Figure 24: Artificially lossy link with 0.1% packet loss probability.

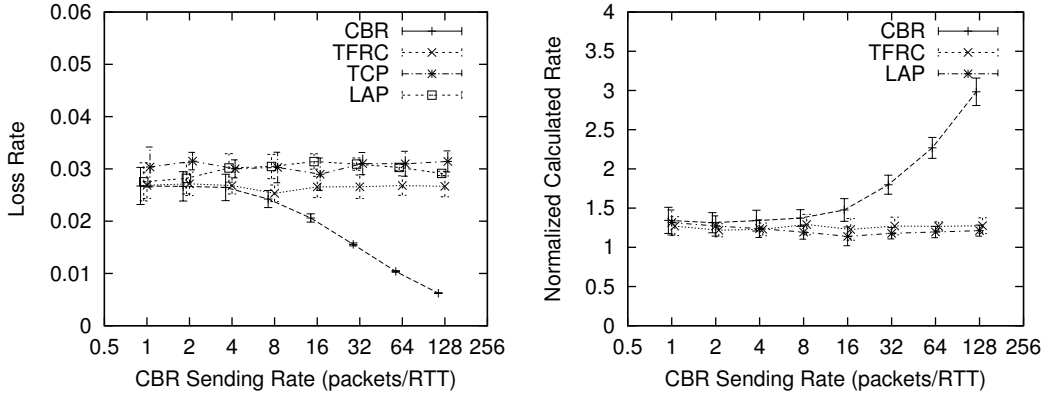


Figure 25: Artificially lossy link with 3% packet loss probability.

round-trip time. However, in Figure 25, it can be seen that the loss event rate of the CBR starts to sink already at eight packets per round-trip time. LAP, however, accurately measures the congestion over the whole span of CBR sending rates.

Interestingly, we also observed that, under certain conditions, LAP follows the CBR and, under other conditions, it follows TFRC. At low CBR sending rates and low degrees of statistical multiplexing both, LIP and LAP, have troubles estimating the congestion. In such conditions, LAP performs equally to LIP, i.e., LAP follows the CBR. However, as the sending rate or degrees of statistical multiplexing increases, LAP starts to follow TFRC, while CBR takes more distance. This can be seen particularly well in Figure 26.

The same pattern of LAP following CBR at low CBR sending rates and then following TFRC at higher CBR sending rates can be seen under higher degrees of

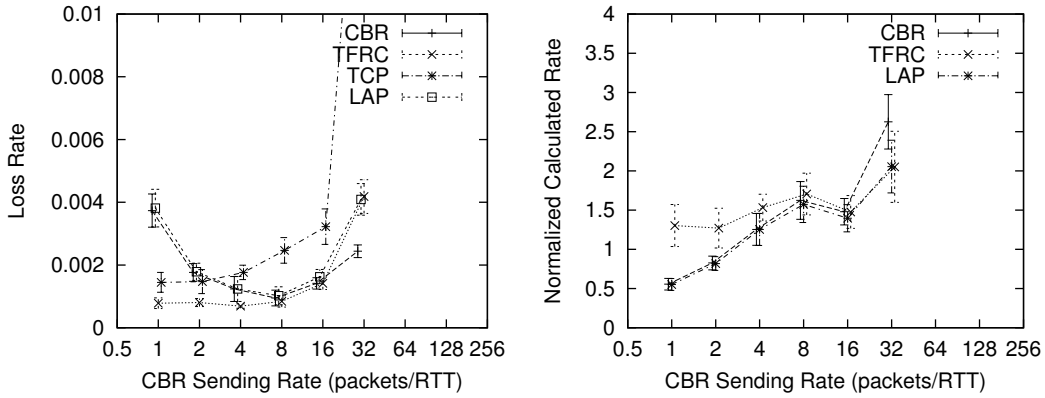


Figure 26: Shared link with one TCP flow.

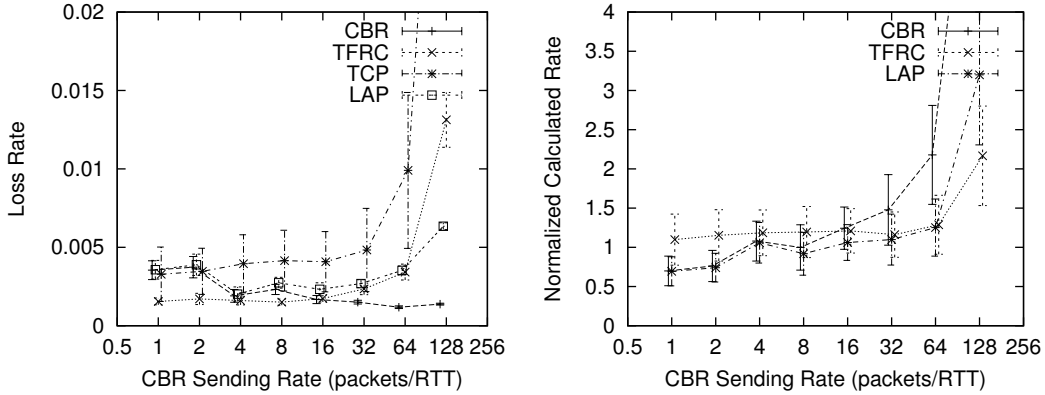


Figure 27: Shared link with eight TCP flow.

statistical multiplexing. Figure 27 shows the behavior with eight competing TCP sources. The pattern is not as clear and radical as with only one competing TCP source.

5.2.4 Open Issues

We have shown how our modified loss even rate algorithm, LAP, accurately measures congestion under variety of conditions. In all simulated cases, it performs at least as well as the original LIP. In many cases, it outperforms LIP under open-loop control scenarios. There are, however, still unsolved issues.

The TCP-model being used assumes packet loss rate and Drop-tail queues. Drop-tail queues, unfortunately, don't give the same signal of congestion to all flows

crossing the queue. Instead, it only signals the flow that happens to overflow it. This alone gives different signals to different flows. This problem materialized particularly at low degrees of statistical multiplexing. However, the difficulty of interpreting a congestion signal is not particular to rate controlled flows but is also present in interaction with different flavors of TCP and AQM [Low03].

5.3 Summary

In this Section we have presented our modification to the TFRC loss event rate algorithm. TFRC uses a long-term TCP model as rate estimation equation. We discussed the assumptions of the underlying model and its differences to the TFRC approach. The model expects packet loss rate as a parameter. However, TFRC computes loss event rate and feeds it in the equation. We confirmed that it gives good results under closed-loop scenario.

We discussed the origin of the dependency and traced it to the Loss Insensitive Period (LIP). It ignores packet losses after the initial packet loss triggering a loss event. At sending rates of multiple packets per round-trip time the loss event rate computation becomes dependent of the number of packet per round-trip time, i.e., on the sending rate.

We presented our modification to the LIP part of the algorithm. Instead of ignoring packet losses after the initial packet loss, we aggregate them to constitute a loss impact. The more packets are lost, the stronger the impact. We call our algorithm Loss Aggregation Period (LAP), and it imitates packet loss rate. Using the same simulation set as in the applicability analysis, we showed that, under open-loop control scenario, it generally provides better loss event rate estimates than the original algorithm. We also identified and briefly discussed scenarios where, both the original, and the modified algorithm, have troubles.

6 Subscription Level Management

In receiver-driven layered multicast congestion control the sender emits data to multicast groups called layers, in a cumulative manner (see Section 2.3.1). The cumulative order of layers is predefined and receivers perform congestion control by subscribing and unsubscribing layers according to the congestion in the network. A subset of layers subscribed by a receiver is referred to as *subscription level*.

While a unicast flow can adapt the sending rate with high granularity, in a layered multicast approach the receiver may only choose from relatively coarse grained rates, i.e., from the layers. This leads to a dilemma of fair competition, oscillations and claim of fair share. When the fair rate is between layers, a recipient may only choose from a layer above the fair rate (too high) or from a layer that is less than the fair share (too low). Long-term fair share could be achieved by oscillating between the two, but oscillations are considered bad. Thus, in a receiver-driven layered congestion control approach, there are scenarios where all of the three are tremendously hard to reach.

Performance of a layered multicast congestion control depends much on the management of the subscription level at receivers. We call this *subscription level management*. There are a number of possible strategies to perform this activity.

In this Section we present three different management strategies and discuss their pros and cons. We choose only one of the strategies and judge our decision to do so. We conclude the Section with simulation results.

6.1 Management Strategies

In this Section we discuss three different strategies and analyze their pros and cons. The strategies are: (1) Naïve strategy, (2) Skeptical strategy, (3) Lazy strategy.

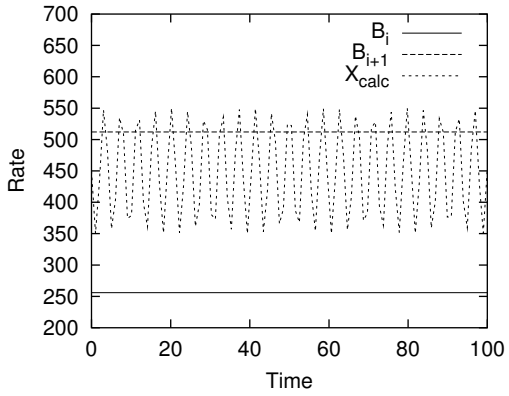


Figure 28: Naïve subscription manager with calculated rate oscillations. (Artificially generated with artificial units)

6.1.1 Naïve Strategy

The naïve strategy is the most simple, as already suggested by the name. It makes decisions directly applying the calculated rate.

Assume that the receiver under consideration has a subscription level of i, i.e., has the layer i as the highest subscribed layer. Then, the total data rate at that layer is B_i. Whenever the calculated rate X_{calc} becomes greater than next layer rate B_{i+1}, layer $i+1$ is subscribed. Similarly, if X_{calc} becomes less than the current layer rate B_i, layer i is unsubscribed.

This strategy is simple to implement but suffers from potential layer oscillation. When the calculated rate X_{calc} oscillates such that peaks cross layer rates B_i and/or B_{i+1}, a join/leave decision is made. The closer the average calculated rate is to a layer rate, the more layer oscillations the strategy is likely experience. Figure 28 illustrates the danger where the peaks are on both sides of the layer rate B_{i+1} causing frequent joins and leaves.

6.1.2 Skeptical Strategy

The Skeptical strategy is a derivation of the naïve strategy with some skepticism introduced. With the skepticism we mean unconfidence of the calculated rate.

Instead of directly subscribing the next layer, or leaving the current layer, we trigger join and leave timers. More precisely, when the calculated rate X_{calc} becomes greater than B_{i+1}, we trigger a join-timer that expires after t_{join} seconds. When the timer fires, layer B_{i+1} is subscribed. Similarly, when X_{calc} becomes less than B_i, we trigger a leave-timer that expires after t_{leave} seconds. When the timer fires, layer B_i is unsubscribed. If the calculated rate drops below B_{i+1} before the join-timer has fired, the join-timer is canceled. Similarly, if the calculated rate raises above the B_i before the leave-timer fires, the leave-timer is canceled.

This strategy has two potential benefits. First, it attempts to prevents frequent layer oscillation in presence of calculated rate oscillations. Second, it allows tuning of aggressiveness (through join-timer) and responsiveness (through leave-timer) separately. The drawback is the difficulty of finding proper values for the timers. If the join-timer is tuned too long, layer $i + 1$ is not subscribed, although average rate would be above B_{i+1} since the join-timer gets canceled before it fires. The same applies to unsubscribing a layer.

6.1.3 Lazy Strategy

The Lazy strategy is a derivation of the Skeptical strategy. While in Skeptical strategy the timers are fixed, in Lazy strategy they are adjusted as a function of the calculated rate distance to the current or next layer.

The objective of Lazy strategy is to stick to the latest decision when possible, and to avoid changes in the subscription level. To do so, we adopt the timers similar to the Skeptical strategy. However, we define a passive zone in which the timers are not triggered at all. The passive zone spans slightly over the current and next layer, i.e., slightly below the currently subscribed layer i and slightly beyond the next layer $i + 1$. Figure 29 illustrates the passive zone.

When the calculated rate enters or exits the passive zone, a timer is triggered. That is, when the calculated rate X_{calc} falls below the passive zone (and thus below the currently subscribed layer i, too), a leave timer is triggered. Similarly,

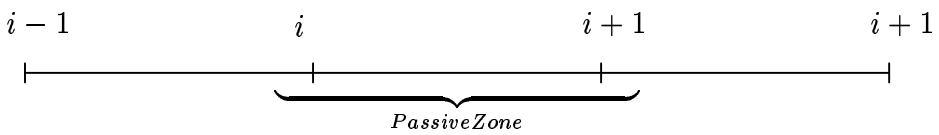


Figure 29: Passive Zone of Lazy strategy.

when the calculated rate gets above the passive zone (and thus above the next layer $i+1$), a join timer is triggered. The timer get canceled according to the same rules of Skeptical strategy.

However, the triggered timers are constantly updated according to X_{calc}. For the update, we introduce aggressiveness and responsiveness functions and use them to derive laziness functions for joins and leaves. The basic idea is to make decisions in a lazy manner, i.e., to postpone the decision. When the calculated rate is only slightly above the next layer, or slightly below the current layer, we try to avoid making decisions.

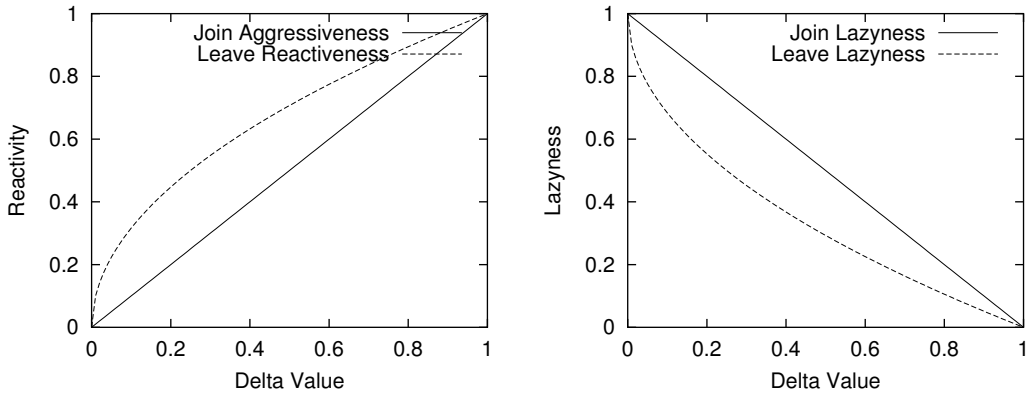
We linearly map the X_{calc} distance to the layer rates B_i or B_{i+1} to get *delta values* in the range $[0, 1]$. That is, if X_{calc} is exactly in the middle of B_{i-1} and B_i, then delta value for leaving is 0.5. Equations 18 and 19 show how to compute the delta values for joining and leaving, respectively.

$$\delta_{join} = \frac{X_{calc} - B_{i+1}}{B_{i+1} - B_i} \tag{18}$$

$$\delta_{leave} = \frac{B_i - X_{calc}}{B_i - B_{i-1}} \tag{19}$$

Figure 30 illustrates reactivity functions, i.e., join aggressiveness and leave reactiveness as functions, of the delta value. The join reactivity functions are $r_{join}(x) = x$ for joining and $r_{leave}(x) = \sqrt{x}$ for leaving. The laziness functions are $l_{join}(x) = 1 - r_{join}(x)$ and $l_{leave}(x) = 1 - r_{leave}(x)$. Thus, the system is more reactive (or less lazy) when it comes to leaving a layer compared to joining a layer. Depending on the application, different functions may be appropriate.

The laziness in making a decision is controlled through a configured delay constant. There is a separate delay constant for joining (d_{join}^{max}) and leaving (d_{leave}^{max}).



(a) Reactivity *(b) Laziness*

Figure 30: Reactivity and laziness as a function of the delta value.

When the calculated rate is only slightly above the next layer, a waiting time close to the maximum waiting time is computed until the layer is joined. The same applies to leaving: when the calculated rate is only slightly below the current layer, a leave delay close to the maximum leave delay is computed. However, if the calculated rate gets far above the next layer, or far below the current layer, the timers are adjusted short. The timers are adjusted by multiplying the maximum join/leave delay constants by the calculated laziness value l_{join} and l_{leave}, respectively. The computations are given in equations 20 and 21.

$$d_{join} = l_{join} \cdot t_{join}^{max} \tag{20}$$

$$d_{leave} = l_{leave} \cdot t_{leave}^{max} \tag{21}$$

This allows us to postpone join and leave decisions when the calculated rate is close to the decision border. However, it still is reactive when the calculated rate falls down suddenly, or rockets up quickly. We haven't investigated what parameters and reactiveness functions provide best results.

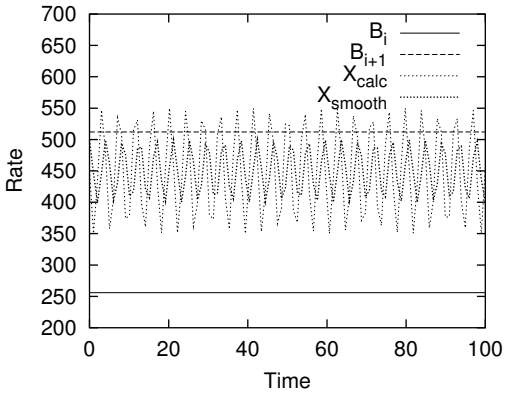


Figure 31: Smooth subscription manager with calculated rate oscillations and the smoothed rate. (Artificially generated with artificial units)

6.1.4 Enhancements

All presented strategies use the calculated rate X_{calc} to manage the subscription level. An option that is applicable to all of the presented strategies is to smooth X_{calc}.

The basic idea is that instead of basing decisions on X_{calc}, the smoothed rate X_{smooth} is used. Smoothing filters high-frequency oscillations providing a more stable rate. Smoothing can be applied to input parameters of the rate calculation function. This is indeed done in TFRC and TFMCC where the round-trip time is smoothed using Exponential Weighted Moving Average (EWMA). The effect of smoothing an oscillating signal is illustrated in Figure 31.

Smoothing can be implemented in several ways. One of the simplest methods is indeed EWMA. In EWMA there is no need to keep a sample of X_{calc} which makes it efficient and easy to implement. The smoothed rate is computed using X_{calc} and a damping factor α as follows:

$$X_{smooth} = (1 - \alpha) \cdot X_{smooth} + \alpha \cdot X_{calc} \tag{22}$$

When the damping factor α is chosen close to 1, the old values of X_{smooth} are changed slowly and new values of X_{calc} affect the X_{smooth} slowly. Thus, the

smaller α, the smoother the X_{smooth}, and the higher α, the more rapidly the smoothed rate follows the X_{calc}.

Another approach to smoothing is to use statistical methods over a history. In such an approach a sample of X_{calc} of size k is saved. Then, sample mean can be computed as follows:

$$X_{smooth} = \frac{1}{n} \sum_{i=1}^{k} X_{calc,i} \tag{23}$$

where k is the sample size. Mean of percentiles can be computed as follows:

$$X_{smooth} = \frac{X_{calc,low} + X_{calc,high}}{2} \tag{24}$$

where $X_{calc,low}$ is the lower percentile, e.g., 20 percentile, and $X_{calc,high}$ is the higher percentile, e.g., 80 percentile. The closer the high and low percentiles are to the median, the smoother X_{smooth} becomes [Jai91]. Basically, having sample data allows much more complex analysis of the history.

6.1.5 Challenges

Although accurate rate estimation allows receivers to optimize the subscription level to a fair level, there are scenarios where rate estimation will simply fail. The equation-based strategy requires packet losses or marked packets, congestion signals, in general, to provide an estimate of fair rate. Without these signals the estimator will not function.

Consider an environment with virtually no competing traffic. Assume that the current subscription level is i and the data rate B_i is less than the bottleneck capacity C. Without losses, as is the case without competing traffic, there will be no congestion signal (because there is no congestion) and the estimator will not provide reliable estimate. The only way to use bandwidth efficiently is to infer the capacity. However, every join-experiment above the capacity inherently leads to congestion. This scenario is illustrated in Figure 32.

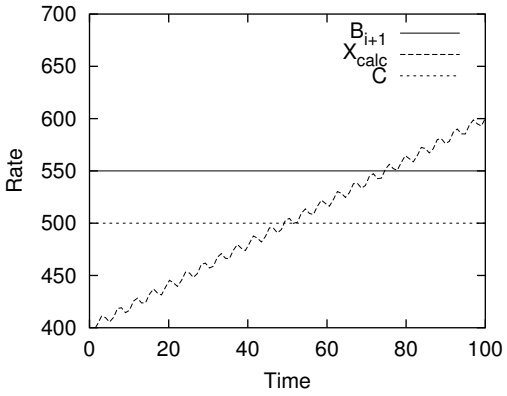


Figure 32: Calculated rate X_{calc} grows beyond capacity C. (Artificially generated)

6.2 Implementation

We decided to implement the Lazy management strategy with smoothing and without the passive zone. We decided to smooth the calculated rate, instead of the the estimated loss rate. To judge the decision and to evaluate different smoothing behaviors, proper simulations would be required. It is left for future work.

We implemented the strategy in NS-2. We implemented a rate control framework designed for unicast and multicast equation-based rate control protocols. We describe only the key points of the implementation and refer the interested reader to `http://www.cs.helsinki.fi/u/colussi/pub/ms-thesis/` for the implementation.

6.2.1 Adaptive Rate Control Framework

To study subscription management strategies using the modified loss estimator we decided to develop a complete framework for NS-2 simulator. We call the developed framework *Adaptive Rate Control Framework* (ARCF). It is targeted to unicast and multicast equation-based rate control protocol research. To verify its applicability to unicast, we implemented the TFRC protocol as defined by [HFPW03].

In ARCF we have decoupled the four components important to rate control and

estimation: (1) round-trip time estimator; (2) retransmission timeout estimator; (3) formula (i.e. the TCP-model providing the rate equation); and (4) loss fraction estimator. In NS-2 all protocol implementations are called agents and they inherit the `Agent` base class. We designed a derived class hierarchy such that sources and sinks where separated. From sources and sinks we then derived unicast and multicast sources. This allowed elegant structuring of shared and individual features.

The manager itself was implemented in unicast and multicast source and sink agents directly. The manager is chosen by setting a *bound* variable `manager_id`. The source or sink then executes common functions as usual, but individual code using `switch/case` statements and this way executing manager code.

6.2.2 LIP/LAP Window

While experimenting our LAP-modification, we discovered interesting packet loss patterns. While TCP sends back-to-back packets and loses all packets following an initial packet loss (because Drop-tail queue is full), a rate controlled flow sends packets in a certain frequency. Assume that the bottleneck capacity is C and that a rate controlled flow transmits packets at rate $R = 2 \cdot C$ and shares the link with nobody. Then, for each dequeued packet two packets arrive to the queue but only one may be enqueued. This results in a pattern where every second packet is lost and every second forwarded.

The above behavior is particular to environments with low degrees of statistical multiplexing. Such losses should be accounted as one loss event in a similar way TFRC does. Thus, we are missing a mechanism to identify the loss pattern: whether a loss event originates from an environment with high level of statistical multiplexing or from an environment with low level of statistical multiplexing. In the former LIP performs well, while in the latter LAP gives better estimates.

To capture losses that are likely due to a Drop-tail queue, we developed a sliding window mechanism to assist the accounting of packet losses. When the packet

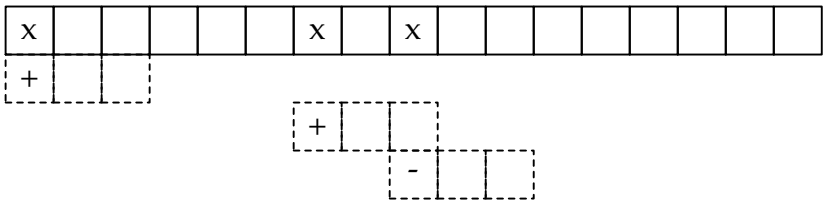


Figure 33: LIP/LAP Window.

losses are grouped, we would like to operate like LIP, that is, to compute the loss event rate as $p_{event} = 1/\Theta_{avg}$. When the packet losses are not grouped, we would like to operate like LAP, i.e., to compute the loss event rate as $p_{event} = \Psi_{avg}/\Theta_{avg}$. We call the sliding window $LIP/LAP - window$, or simply window, and refer to it with W.

Assume that the window size is N. Whenever a packet loss that triggers a new loss event is detected, the loss is accounted and the window is set to the lost packet sequence number S_i. When the next loss in the same round is detected, the lost sequence number S_j is compared to W. If $W + N > S_j$, then the loss is ignored. Otherwise it is accounted as defined by LAP. At each packet loss, W is updated to point to the last lost packet sequence number. Figure 33 illustrates the use of the LIP/LAP window with window size of three packets. The solid boxes represent packets (lost packets marked with x) in the aggregation period and the dashed boxes represent the LIP/LAP Window being updated at packet losses (accounted losses marked with + and ignored losses marked with -). In the Figure three packets are lost, but only two of them are accounted.

In layered multicasting consecutive losses are not always easily detected. Since receivers may subscribe any cumulative subset of the offered layers, the sequence numbers of packets must be meaningful for all receivers. Thus, all layers must implement a separate sequence number flow. A lost packet on layer i is detected earliest when the first out of order packet on layer i arrives. Thus, a lost packet may be detected slightly later in layered multicast scenario than in multicast scenario.

6.2.3 Feedback Management

In unicast communication, the sender and a receiver form a closed-loop control. It means that the receiver feeds back frequent acknowledgments or control messages. The messages provide information of the network path being used, and enables estimation of round-trip time between the sender and a particular receiver.

In multicasting, closed-loop control is not feasible for large groups. The number of potential feedback messages grow linearly with the multicast group size. Unmanaged feedback could become dominant in the network making poor use of resources, or even rendering the multicast group useless. This is technically called *feedback implosion*. There are several possible approaches to feedback management, e.g., exponentially distributed timers [NB98].

Estimation of round-trip time is a necessity for estimating a TCP-compatible data rate. In addition to that, feedback may provide information for the sender to optimize the data parameters according to receiver reports. Such optimization depends on the application.

In our context feedback is mostly required to enable estimation of round-trip time. We implemented a relatively simple management algorithm and do not expect it to perform optimally or to scale to very large groups.

Our algorithm is driven by the source in co-operation with receivers. The source is configured to have a target feedback rate R_{target} as feedback messages per second. The sender measures at regular intervals (currently hard-coded to 5 seconds) the feedback arrival rate $R_{current}$. If the current arrival rate is greater than the target rate, then a new interval is computed

$$t_{fb} = \frac{R_{target}}{R_{current}}$$

subject to a maximum increase of 10%. If the calculated rate is less than the target rate, the interval is decremented 10%.

The feedback interval t_{fb} is injected in all packets emitted to the base layer. Since base layer packets are received by all participants, it is by definition received by all participants. The receivers then use the given feedback interval to adjust feedback timers. Our current version is simplified such that the timer is updated only when the receiver sends feedback. A robust implementation would constantly adjust the timer according to the given interval.

6.2.4 Slow-Start

We showed in Sections 4.3 and 5.2 how the calculated rate remains below the fair share when the degree of statistical multiplexing and the sending rate are low. In environments with low degree of statistical multiplexing this poses a challenge. At startup, when layers are expected to be gradually subscribed, the calculated rate would prevent the receiver from subscribing further layers (in the beginning the data rate is low until additional layers are subscribed).

To avoid this problem we implemented a slow-start phase which attempts to bring the receiver to the equilibrium at startup. A receiver always starts in the slow-start and increases the subscription level until the equilibrium has been reached. The convergence speed can be adjusted with a parameter *slow-start interval* which we refer to as I. It is the time in seconds that must elapse from previous join before performing the second join. The delay applies to all layers, i.e., independent which layer is being joined, the the delay is always constant.

The interval is also used to adjust a *deaf-period*. Deaf-period means the time, after slow-start join, during which the calculated rate is not consulted. The deaf-period is proportional to the layer bandwidth

$$D_i = I \cdot \frac{B_0}{B_i}$$

where D_i is the deaf-period after joining layer i, B_0 is the base layer bandwidth and B_i is the cumulative bandwidth on layer i. Once the deaf-period has elapsed, if the calculated rate is below B_i, then the slow-start terminates and normal op-

eration is resumed. Thus, the deaf-period becomes shorter as the cumulative bandwidth increases.

In our simulations this slow-start algorithms appears to give fairly good results. However, a thorough analysis would be required to make more general conclusions of its performance. We expect it to be a good start for optimization and further study. Especially the deaf-period requires further study. Although it is directly adjusted by layer rate ratio, and, thus, not really dependent on the layer granularity, it might show bad performance when the layer granularity is set to linear instead of exponential.

6.2.5 Flow Statistics

We described smoothing enhancements in Section 6.1.4. We implemented a C++ component, called *FlowStat* to perform statistical processing on a data flow. The component utilizes `vector` and `map` templates to keep track on the flow data.

When the FlowStat is instantiated, it is initially empty. For each event, e.g., packet arrival, the current calculated rate is added in the statistics component. The component keeps all values in FIFO order such that when the maximum size has been reached, each addition causes a remove of an old item in FIFO order. It also keeps all values sorted such that the median value is in the middle of a vector. It follows that any percentile can be found in constant time.

Other statistical measures, such as mean, is possible to compute in constant time with additional bookkeeping. For example, for each addition, increment a sum variable by the new value. For each remove, decrement the sum variable by the removed value. Then, the sum would reflect the sum of all stored items. As the component is aware of the number of items n in the component, through the vector template, it can compute, at any time, the mean by $\bar{x} = sum/n$.

Similar bookkeeping would enable tracking of other statistical measures, such as variance or coefficient of variation. We have only implemented the sorting

of items. We search for percentiles and use the mean of percentiles as average measure. It is worth mentioning that the bookkeeping approach poses a challenge when applied to floating point values. Computation using floating point values is inherently inaccurate. There is potential danger that some bookkeeping methods could produce inaccurate results over a long flow. Then, some mechanism would be require to refresh the bookkeeping by recomputing from the currently stored sample.

6.3 Simulation Results

In this Section we present the simulation results obtained with our implementation of the Lazy strategy. For comparison, we ran the simulation set also with the Receiver-Driven Layered Congestion Control (RLC) (see Section 3.2.2) protocol with default settings. We configured same exponentially distributed layers for both protocols with base layer of 128 kbit/s. We were mainly interested in the decision process and simulated with a zero leave latency.

A multicast congestion control protocol should be tested using multiple topologies and scenarios, such as presented in [BHH+00]. Due to space and time constraints we restricted our simulations to a small subset of the required scenarios. We call our implementation *Equation-based Layered Multicast* (ELM).

We used a tree topology of depth three in all simulations. The topology is illustrated in Figure 34. The default propagation delay on links was 10 ms. Links that were not bottleneck links had a capacity of 100 Mbit/s and a queue large enough not to affect the flows. Bottleneck queues were configured to a different bandwidth and the queue was set to twice the bandwidth-delay product, as recommended by [BHH+00]. All flows sent 1000 byte packets including IP, UDP and TCP headers. We repeated each simulation 15 times to collect data to compute 95% confidence intervals for the mean values. Using the mean throughputs, we computed fairness index (see Section 2.3.2) [JCH84]. All trace figures are from the first replication of a simulation.

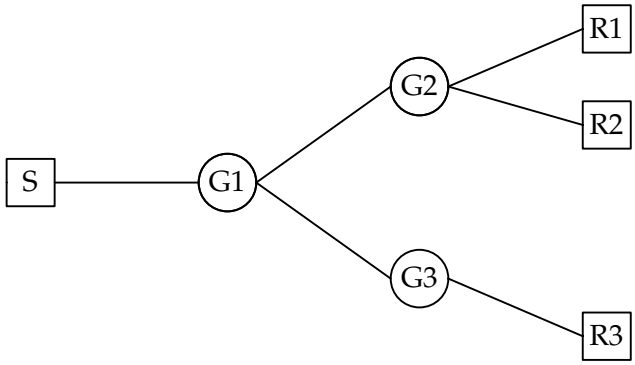


Figure 34: Tree topology of depth three.

Table 2: Simulation Factors and Levels.

Simulation Name	**G1**	**G2**	**R1**	**R2**	**R3**
1. Naïve vs. Lazy	1%	3%	1%	0%	0%
2. Artificial Congestion	1%	3%	1%	0%	0%
3. Congestion due to Competition	-	-	3 Mbps	6 Mbps	12 Mbps
4. Heterogeneous Delay	1%	1%	250 ms	50 ms	500 ms
5. Transient Congestion	-	-	3 Mbps	6 Mbps	12 Mbps

We run multicast receivers and TCP receivers in all receiver nodes (R1, R2 and R3) and one multicast sender and all TCP senders in the sender node (S). In different scenarios we varied the link properties and number of competing TCP sources. The link configurations for the five simulations are summarized in the Table 2. With the simulations we attempt to highlight the strengths and weaknesses of our modified rate estimation algorithm in conjunction with the Lazy strategy. We had the join and leave delay constants set to 25 seconds. It is a long time but we don't currently have any better knowledge of good values. A large value is also likely to show weaknesses of the laziness functions if it is to perform poorly. We designed the laziness algorithm to allow quick reaction to large changes and expect to see it reaction fairly quickly.

6.3.1 Naïve vs. Lazy

In our implementation we focused on the Lazy strategy. However, to get insight to its performance, it is important to compare it to other strategies. In this simulation we used artificially lossy links with homogeneous delays and run one Naïve and one Lazy instance in each receiver node. We simulated the instances in two different simulations. In the first simulation, a Naïve instance was configured in each receiver node with a TCP/Sack. In the second simulation, a Lazy instance was configured in each receiver node with a TCP/Sack.

We expected to see more oscillations in the Naïve instances than in the Lazy instances. The closer the path fair rate is to a layer, the more the Naïve may oscillate. The results show more oscillations in the Naïve instance and TCP mean throughput such that the confidence intervals overlap. Thus, we can't make statistical statements about their differences.

The Naïve algorithm implemented a slow-start that had a join-delay of a multiple of round-trip time. Hence, it performed joins much quicker than the Lazy and this can be seen in the beginning of the plots.

Figure 35 shows the Naïve and Lazy at node R1. The packet loss probability on the path is 2%. Mean TCP throughput, with the Naïve instance, is 927 kbps, and with the Lazy instance 909 kbps. The mean throughput of the Naïve instance is 649 kbps and of the Lazy instance 785 kbps. Thus, the fair rate on the path remains below 1024 kbps which is the layer closest to the fair rate. Subscribing this layer discriminates TCP. However, the discrimination is relatively small. The Naïve instance does occasionally subscribe the 1024 kbps layer but leaves it quickly leading to layer oscillations. The Lazy instance keeps the subscription level because the discrimination is so small. The Naïve instance makes around 45 subscription level changes (including slow-start) while the Lazy makes only around 12.

Figure 36 shows the instances in node R2. The path packet loss probability is

Table 3: Fairness index in simulation 1.

Protocol	**R1**	**R2**	**R3**
Naïve	0.9689	0.9756	0.9745
Lazy	0.9914	0.9769	0.9825

1% which leads to a higher TCP throughput. The TCP mean throughput, with Naïve is 1416 kbps and, with Lazy 1438 kbps. Naïve mean throughput is 1033 kbps and Lazy mean throughput is only slightly more, 1059 kbps. The TCP mean throughput is almost in the middle of 1024 kbps and 2048 kbps layers leading to a more stable operation. Naïve makes now slightly over 30 subscription level changes while Lazy makes less than 10.

Figure 37 shows the instances in node R3 where the packet loss probability is 3%. This results in a lower TCP throughput. The mean TCP throughput, in both simulations, was 662 kbps. Naïve mean throughput was 479 kbps and Lazy mean throughput was 508 kbps. Again, Lazy achieves more stable operation and makes less than 10 subscription level changes while Naïve makes around 34 subscription level changes.

We have summarized the fairness indexes in Table 3. The table shows that Lazy provides better fairness to TCP than Naïve under steady state operation. In the subsequent sections we refer to Equation-based Layered Multicast (ELM) meaning the Lazy strategy, and compare it to RLC.

6.3.2 Artificial Congestion

RLC performs congestion control by reacting to packet losses. By generating artificial congestion we can easily observe the protocol behavior under a certain level of congestion. In this simulation we run one instance ELM, RLC and TCP in all receiver nodes simultaneously. For this reason the ELM and RLC have been plotted with the same TCP flow.

Figure 38 shows ELM and RLC at node R1 where the packet loss probability is

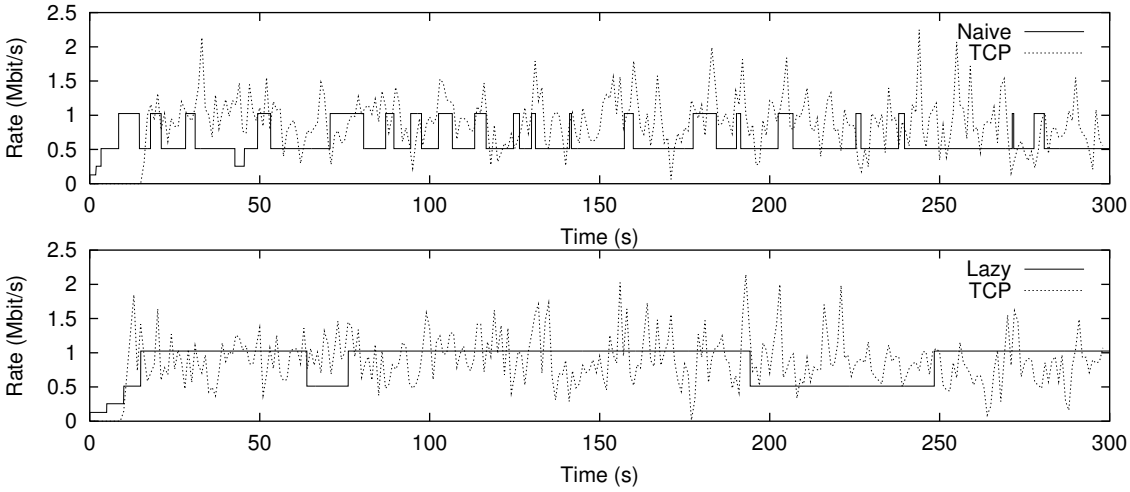


Figure 35: Simulation 1 at node R1. Path loss probability 2%.

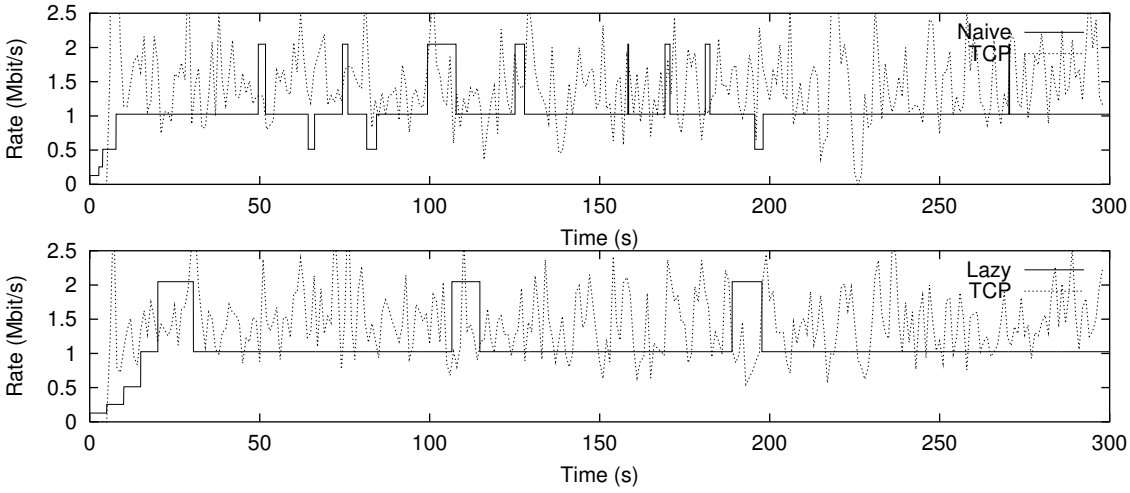


Figure 36: Simulation 1 at node R2. Path loss probability 1%.

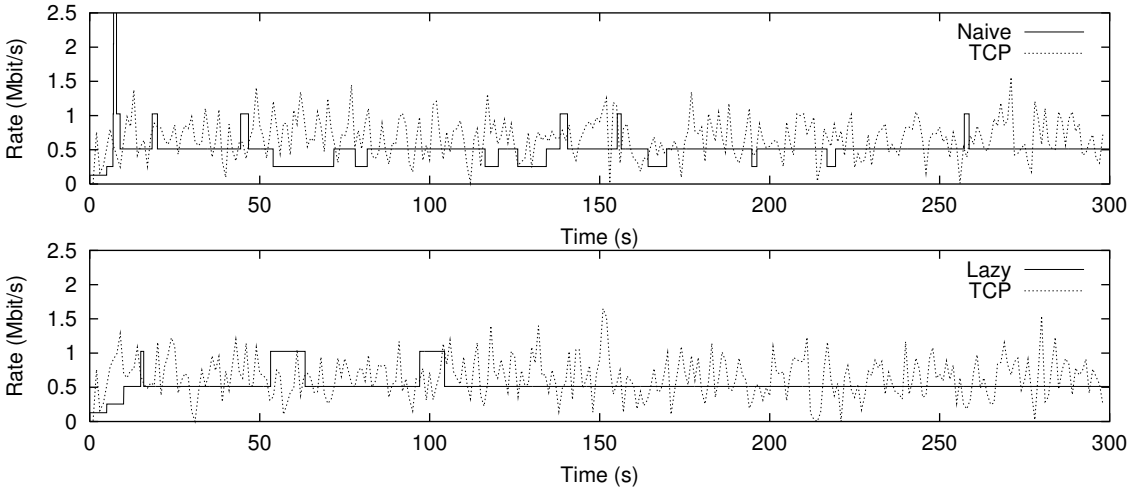


Figure 37: Simulation 1 at node R3. Path loss probability 3%.

2%. TCP mean throughput is 912 kbps, ELM mean throughput 783 kbps and RLC mean throughput is 470 kbit/s. RLC grabs significantly less bandwidth than TCP on the path. The reason to such behavior is that RLC operates on losses only and is not aware of the round-trip time. The smaller round-trip time, the more bandwidth TCP gets. In this scenario the round-trip time was 60 ms. In this configuration, ELM gets a round-trip time estimate, on the average, every three seconds. Thus, it has quite accurate estimate and can well support the rate estimation. Its fair share, however, remains below TCP because the 1024 kbps layer is too much over the fair share and it conservatively subscribes it. ELM makes, on the average, 12 subscription level changes while RLC makes around 430 changes.

Figure 39 shows ELM and RCL at node R2 where the packet loss probability is 1%. The TCP mean throughput is 1429 kbps and almost in the middle of the 1024 kbps and 2048 kbps layers. It leads to a stable operation of ELM and a mean throughput of 1076 kbps which is very close to the 1024 kbps layer. RLC mean throughput remains as low as 641 kbps. Again, ELM makes less than 10 subscription level changes while RLC makes way beyond 300 layer changes.

Figure 40 shows ELM and RLC at node R3 where the packet loss probability is 3%. Mean TCP throghput is 668 kbps which is well above the 512 kbps layer. ELM mean throughput is 508 kbps which is slightly below the 512 kbps layer. RLC throughput is 400 kbps. ELM makes only about 7 layer changes while RLC makes almost 500 layer changes which sums up to more than one change per second.

We have summarized the fairness indexes in Table 4. The table shows that ELM provides better fairness to TCP than RLC under artificially congested link. The fairness index of RLC appears to be slightly lower than what the authors have measured under shared link with Drop-tail and RED queuing policy [VCR98]. Artificial congestion appears to be hard for RLC since the back-off doesn't have any effect on the artificial congestion which it expects.

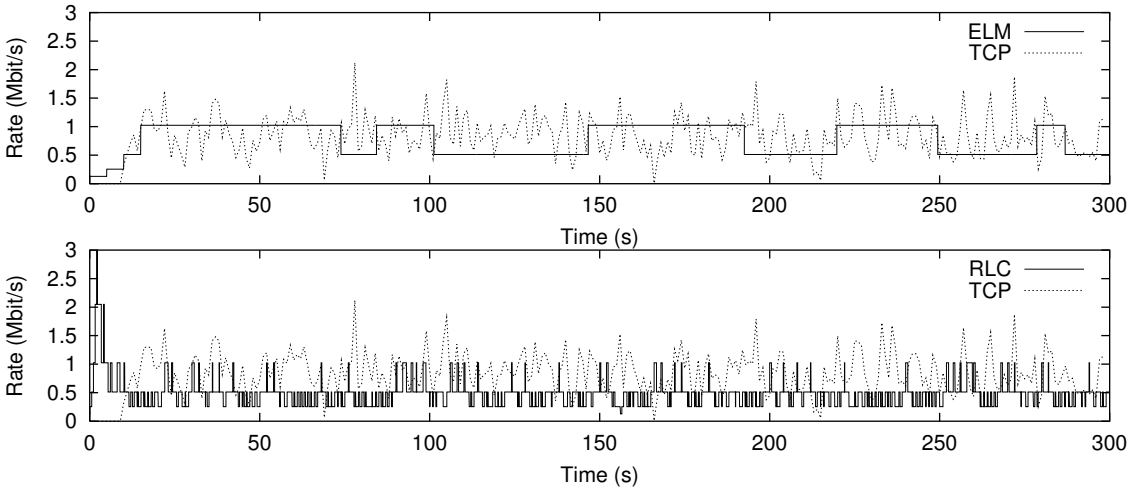


Figure 38: Simulation 2 at node R1. Path loss probability 2%.

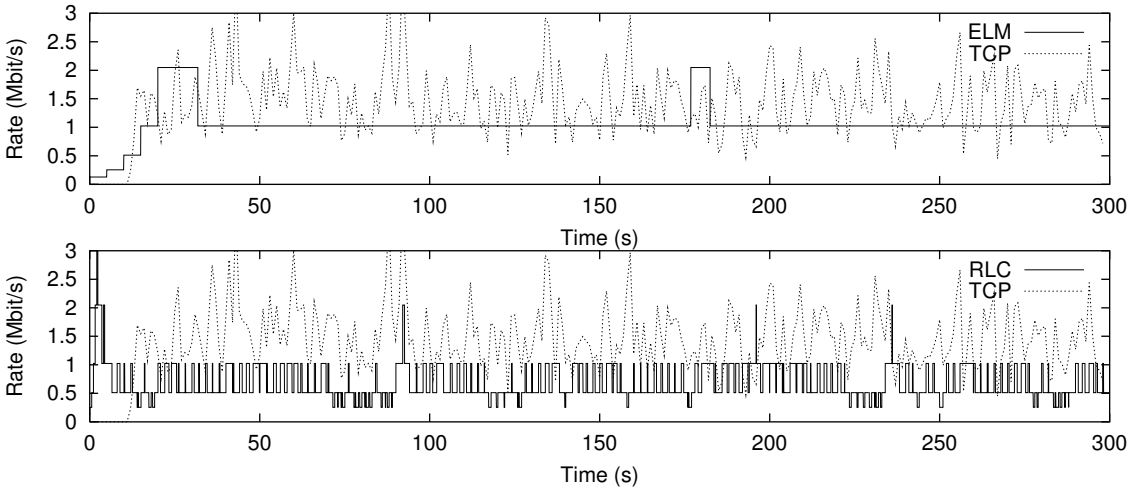


Figure 39: Simulation 2 at node R2. Path loss probability 1%.

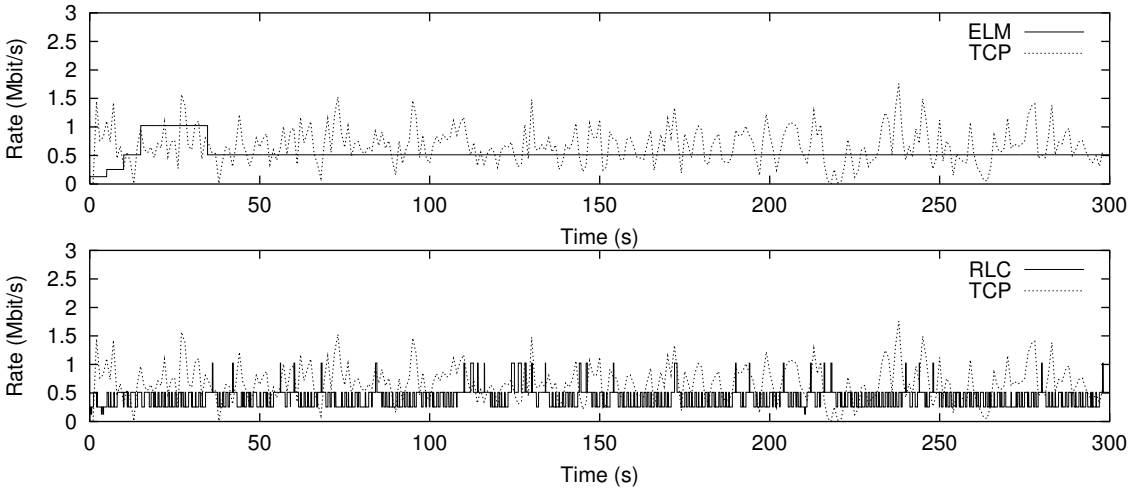


Figure 40: Simulation 2 at node R3. Path loss probability 3%.

Table 4: Fairness index in simulation 2.

Protocol	R1	R2	R3
ELM	0.9921	0.9801	0.9814
RLC	0.9072	0.8733	0.9406

6.3.3 Congestion due to Competition

With the simulation setup of artificial losses we attempted to capture protocol behavior under circumstances where the congestion occurs in a link with very high degree of statistical multiplexing. We now turn to environments with low degree of statistical multiplexing. In our configuration we have one ELM, RLC and four TCP at each receiver node. The multicast senders start at 10 seconds and the TCP connections start randomly in the window of 0 to 20 seconds.

The link bandwidths are configured such that node R1 has 3 Mbps bottleneck, R2 has 6 Mbps bottleneck, and R3 has 12 Mbps bottleneck. All bottleneck queues were configured to twice the bandwidth-delay product. The round-trip propagation delay to all nodes was 60 ms. To prevent ELM and RLC from affecting each other, we run the simulation twice: first with ELM and TCP, and then with RLC and TCP.

Figure 41 shows ELM and RLC with TCP at node R1 where the bottleneck bandwidth is 3 Mbps. With ELM, the TCP throughput is 632 kbps and, ELM throughput 550 kbps which is slightly above the 512 kbps layer. With RLC, the TCP throughput is 645 kbps and, RLC throughput 478 kbps. Particularly interesting to observe how well ELM slow-start converges to the 512 kbps layer. Almost all of the 15 replications converge in 25 seconds making only one layer subscription too much to the 1024 kbps layer. ELM performs about 9 layer changes while RLC performs almost 300 changes.

Figure 42 shows ELM and RLC with TCP at node R2 where the bottleneck bandwidth is 6 Mbps. With ELM, the TCP throughput 1309 kbps which is above the 1024 kbps layer. However, ELM throughput is nevertheless only 790 kbps. With

RLC, the TCP throughput is 1353 kbps and RLC throughput is 597 kbps. ELM does not attempt to join the 1024 kbps layer more aggressively than in R1. The calculated rate oscillates heavily preventing the Lazy strategy to joining and keeping the 1024 kbps layer.

Figure 43 shows ELM and RLC with TCP at node R3 where the bottleneck bandwidth is 12 Mbps. With ELM, the TCP throughput is 2518 kbps and, ELM throughput is 2231 kbps which is above the 2048 kbps layer. The fact that the equilibrium rate is so high (above 2048 kbps layer) shows a weakness in the slow-start algorithm. In slow-start, joins are delayed always with the constant configured time. However, the deaf-period is proportional to the ratio of the subscribed layer and the layer bandwidth ratio. The higher the rate, the shorter the deaf-period. The result is that at high rates the deaf-period becomes really short. When the deaf-period is too short, the calculated rate doesn't have time to converge to the new layer, although the equilibrium was above the layer. This makes ELM exit the slow-start prematurely leading to a very slow convergence.

In node R3, the TCP with RLC has a throughput of 2768 kbps and the RLC has a throughput of 1113 kbps which is above the 1024 kbps layer. RLC oscillates less than so far observed: in this simulation RCL makes around 150 layer changes. The higher equilibrium bandwidth could be a reason to it. For higher bandwidth the bottleneck queue is larger. Also, in congestion avoidance, TCP grows its rate one packet per round-trip time. Then, the queue limit takes longer to reach. Since the queue limit is reached less frequently, also RLC has more time to operate without losses which shows as fewer oscillations.

We have summarized the fairness indexes in Table 5. In this simulation ELM shows higher fairness index values than RLC.

6.3.4 Heterogeneous Delay

TCP throughput is inversely proportional to the round-trip time. RLC reacts only to packet losses and doesn't take the round-trip time into account. This can

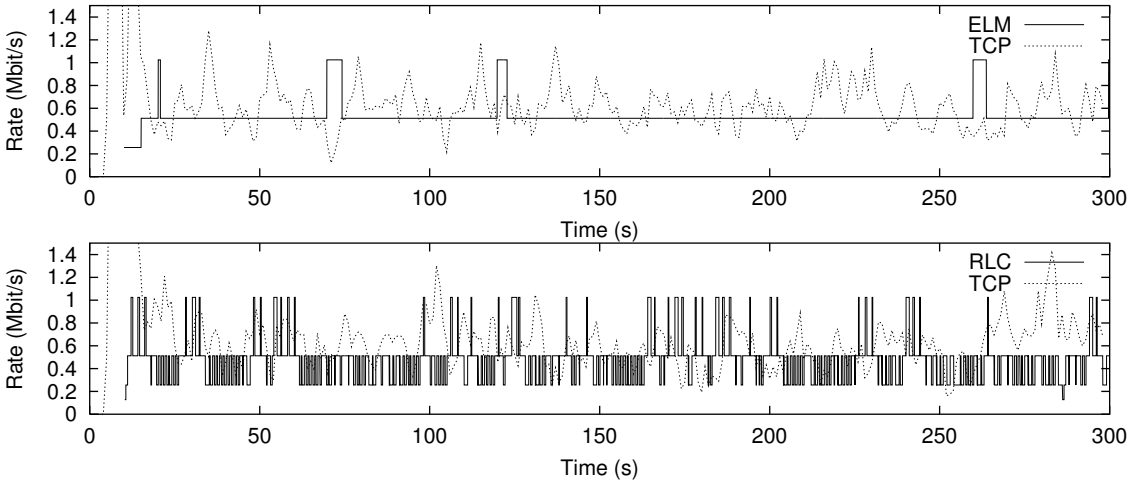


Figure 41: Simulation 3 at node R1. Theoretical fair share 0.6 Mbps.

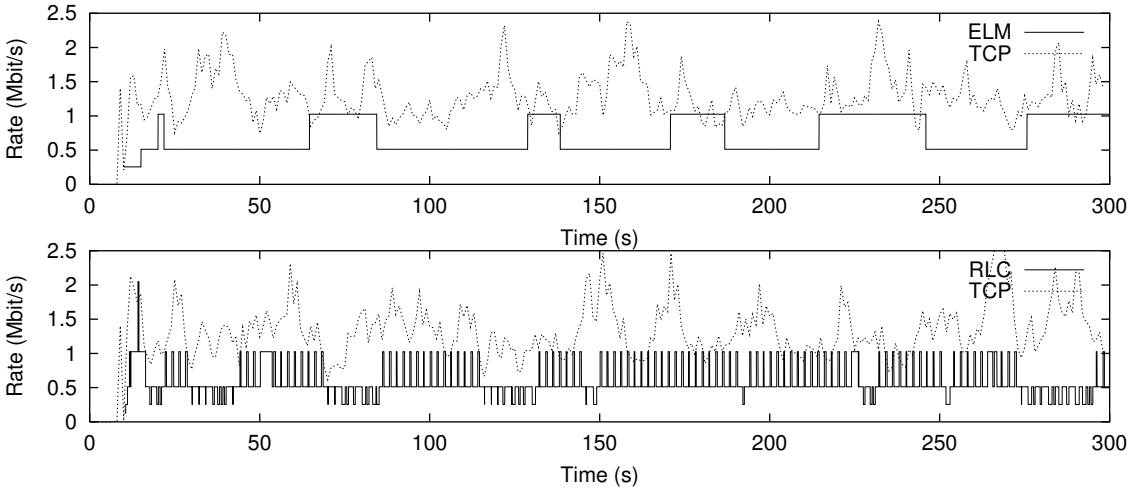


Figure 42: Simulation 3 at node R2. Theoretical fair share 1.2 Mbps.

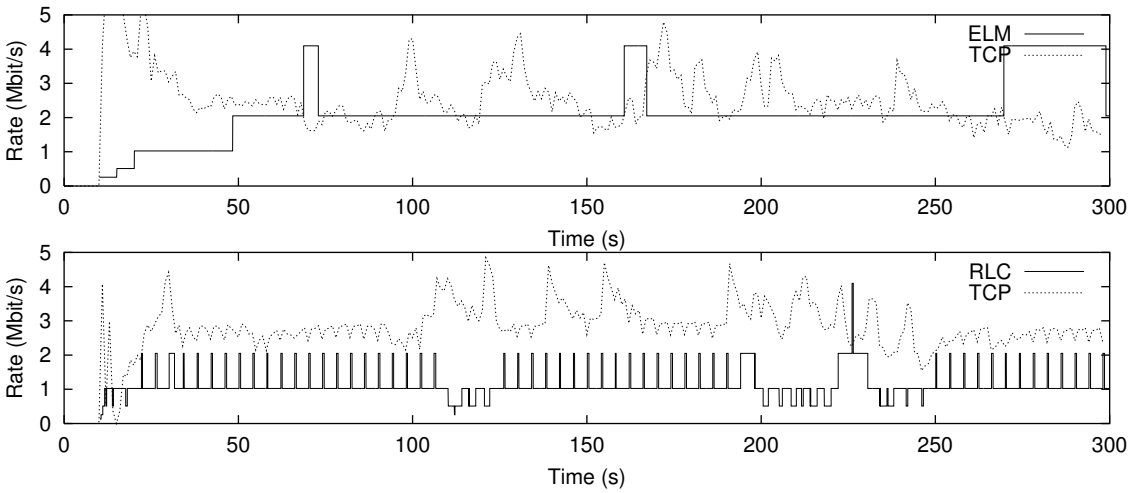


Figure 43: Simulation 3 at node R3. Theoretical fair share 2.4 Mbps.

Table 5: Fairness index in simulation 3.

Protocol	**R1**	**R2**	**R3**
ELM	0.9967	0.9691	0.9961
RLC	0.9871	0.9387	0.9311

lead to unfair bandwidth allocation between RLC and TCP - or with any TCP-compatible protocol. ELM uses an equation-based approach where the equation is used to estimate a TCP-compatible data rate. ELM sends feedback to the sender and estimates feedback using the timestamps in the feedback messages.

In this simulation we configured different round-trip times for all receiver nodes by configuring link propagation delays. The link S-G1 propagation delay was configured to 10 ms, G1-G2 to 10 ms, G1-G3 to 120 ms, G2-R1 to 105 ms, G2-R2 to 5 ms and G3-R3 to 120 ms. Node R1 experiences a round-trip time of 250 ms, R2 experiences a round-trip time of 50 ms and R3 experiences a round-trip time of 500 ms. The delays are illustrated in Figure 44. The round-trip times are indicative and the queuing delay is ignored. The configured round-trip times are widely distributed and allow us to observe the protocol behaviors and capture potential unfairness issues originating from the heterogeneous delays. In other simulations the round-trip time to all nodes has been 60 ms. In addition to delay, we configured artificially lossy links to nodes G1 and G2 with packet loss probability of 1%. We simulated ELM with one TCP per node (R1, R2 and R3) first, then RLC with one TCP per node.

Figure 45 shows ELM and RLC at node R1 with round-trip time of 250 ms. The TCP throughput in the plot with ELM is 342 kbps and the ELM throughput is 261 which is slightly more than the 256 kbps layer. The TCP throughput in the plot with RLC is 362 kbps and RLC throughput is 667 kbps which is almost double the TCP throughput. RLC is clearly discriminating TCP in this case. Since ELM measures the round-trip time and has a quite accurate estimate, it manages to compute a fair rate estimate and choose a proper layer.

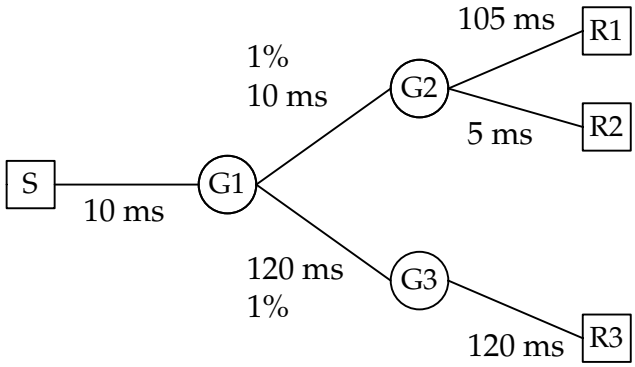


Figure 44: Topology of simulation 4.

Figure 46 shows ELM and RLC at node R2 with round-trip time of 50 ms. The TCP throughput in the plot with ELM is 1726 kbps and the ELM throughput is 1417 kbps. The TCP throughput in the plot with RLC is 1703 kbps and the RLC throughput is 643 kbps. In this case RLC is not taking its fair share. ELM, however, keeps a fair subscription level, although it several times attempts to join the unfair layer of 2048 kbps.

Figure 47 shows ELM and RLC at node R3 with the largest round-trip time, 500 ms. With such a long delay TCP can be expected to get a fairly small share of the bandwidth. The TCP throughput in the plot with ELM is only 176 kbps and ELM throughput is 132 kbps. That is only a bit above the base layer at 128 kbps. The TCP throughput is 179 kbps and the RLC throughput is 637 kbps.

These simulations have showed that with heterogeneous delays RLC behave rather unfairly to TCP, or any other TCP-compatible protocol. The results also show that ELM manages to, provided a representative round-trip time estimate, keep a subscription level that is fair to TCP. We have summarized the fairness indexes in Table 6. Also, ELM avoids layer changes and does, on the average, in worst case only about 14 layer changes. In the same time RLC changes layer around 300 times.

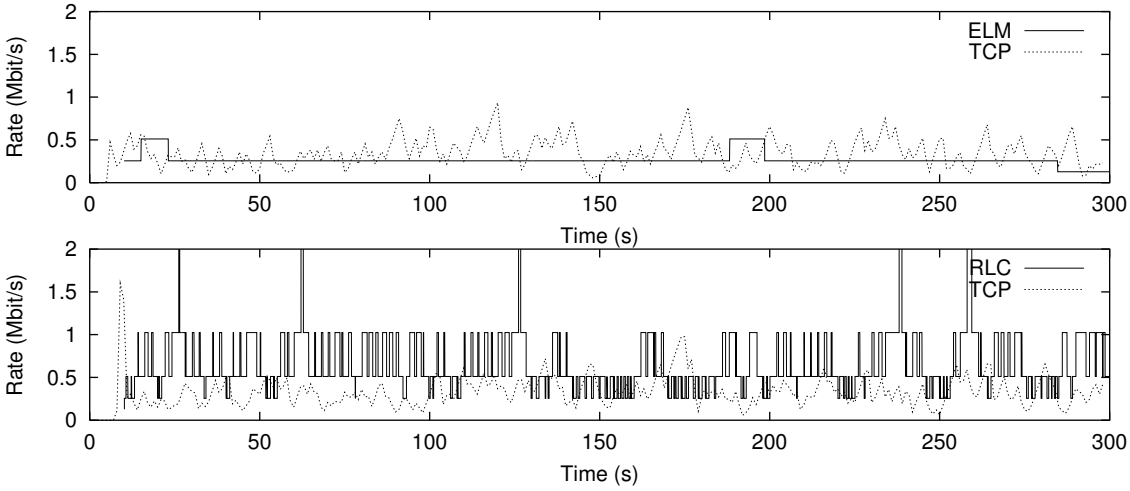


Figure 45: Simulation 4 at node R1. Round-trip time 250 ms.

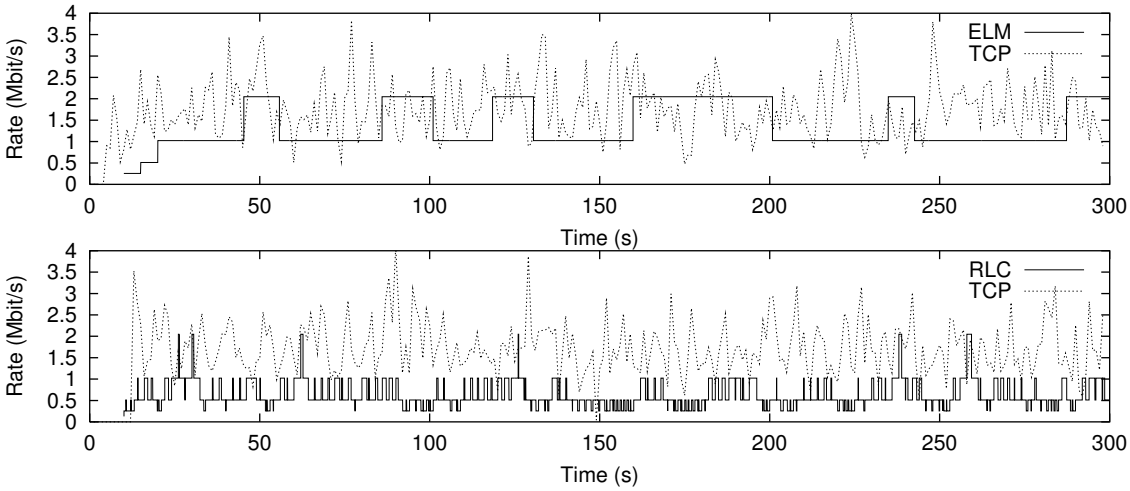


Figure 46: Simulation 4 at node R2. Round-trip time 50 ms.

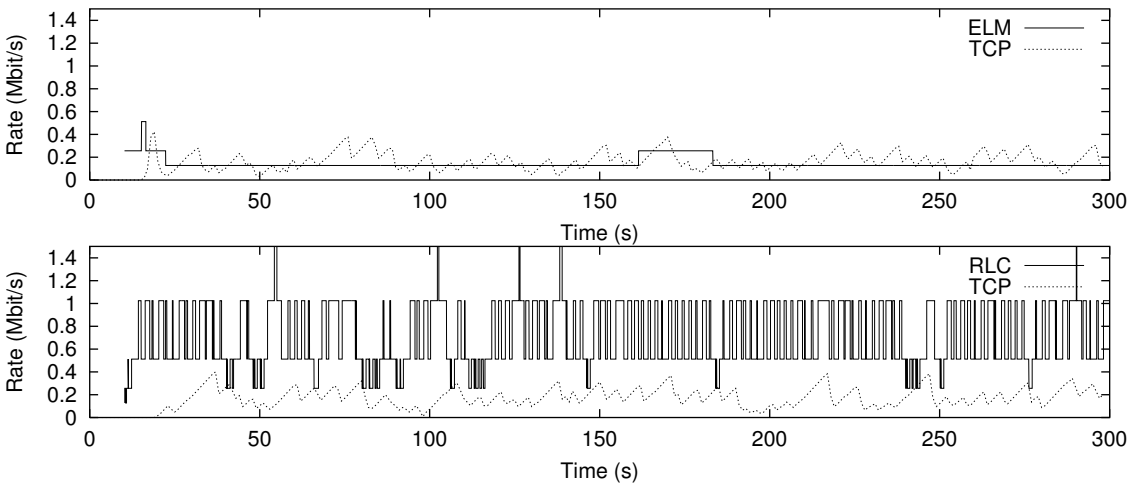


Figure 47: Simulation 4 at node R3. Round-trip time 500 ms.

Table 6: Fairness index in simulation 4.

Protocol	R1	R2	R3
ELM	0.9816	0.9892	0.9784
RLC	0.9189	0.8304	0.7615

Table 7: Additional TCP flow times.

Node	Start time	End time
R1	100	140
R2	200	220
R3	230	240

6.3.5 Transient Congestion

So far we have investigated only static scenarios where all flows have static environment. A dynamic scenario has changing conditions - other changes than what the flows do themselves - which affect the flows. We generate disturbance to the steady system by doubling the number of TCP connection between the sender node and receiver nodes at three different times. We then observe how the multicast algorithms react to the disturbance.

We configured the receiver node R1 to have a bottleneck bandwidth of 3 Mbps, R2 to 6 Mbps and R3 to 12 Mbps. All links were set to propagation delay of 10 ms which sums to 60 ms round-trip propagation delay for all nodes. All nodes ran one ELM, one RLC and, four TCP connections. In addition to the these flows, we started four additional TCP flows at different times according to Table 7. Ideally, ELM would drop the highest layer.

Figure 48 shows ELM and RLC at node R1 where the additional TCP flows start at time 100 seconds and cease at 140 seconds. The additional flow duration is 40 seconds. The theoretical fair share with one multicast flow and four TCP flows is 600 kbps. When the four additional TCP flows start, the theoretical fair share is 333 kbps. Thus, the multicast flow should definitely leave the 512 kbps layer.

Table 8: Fairness index in simulation 5.

Protocol	**R1**	**R2**	**R3**
ELM	0.9331	0.9655	0.9858
RLC	0.9249	0.9387	0.9311

However, from the 15 replications some ELM flows require more than 10 seconds to leave the 512 kbps layer. And some even decide to join the 512 kbps layer again before the additional flows have ceased.

Figure 49 shows ELM and RLC at node R2 where the additional TCP flows start at time 200 and cease at 220 seconds. The additional flow duration is 20 seconds. On a 6 Mbps bottleneck bandwidth the theoretical fair rate with one multicast and four TCP flows is 1200 kbps. The fair layer is then at 1024 kbps. With the additional four TCP flows, the fair rate falls down to 666 kbps where the fair layer is 512 kbps. ELM flows in all 15 replications drop the 1024 kbit layer in about five seconds after the TCP flows start. Almost all ELM flows are much more conservative to join the layer again after the TCP flows have ceased. For some flows it takes more than 20 seconds.

Figure 50 shows ELM and RLC at node R3 where the additional TCP flows start at time 230 and cease at 240 seconds. The additional flow duration is 10 seconds. In this case ELM flows react, again, in about 5 seconds and leave the 1024 kbps layer. Joining layers again is slower. Some flows join 1024 kbps layer in 25 seconds after the TCP flows cease, and other flows join after 50 seconds.

We have summarized the fairness indexes in Table 6. These simulations results suggest that the chosen laziness function and delay constants should be investigated in detail. However, the basic concept shows fairly good behavior under a wide range of conditions.

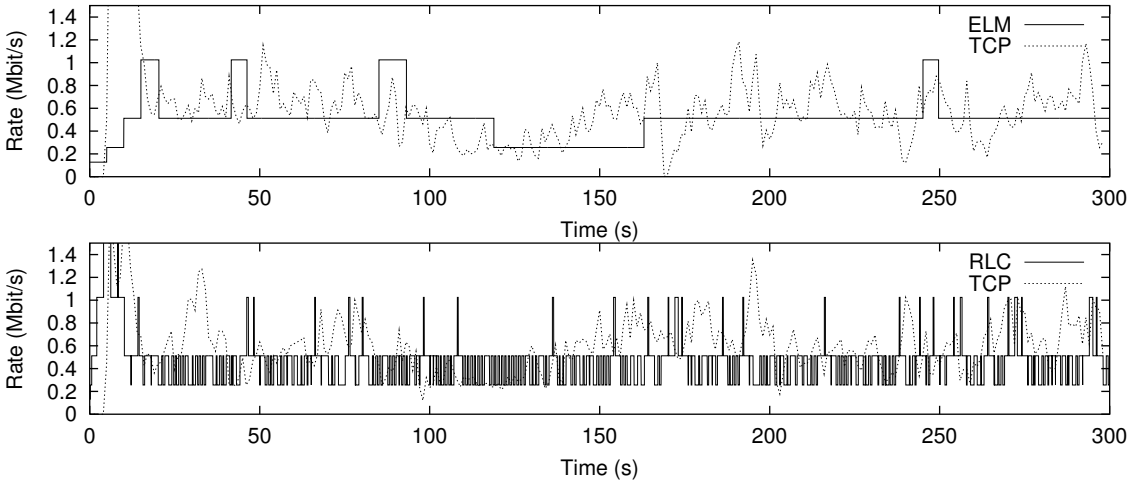


Figure 48: Simulation 5 at node R1. TCP burst 100 to 140 seconds.

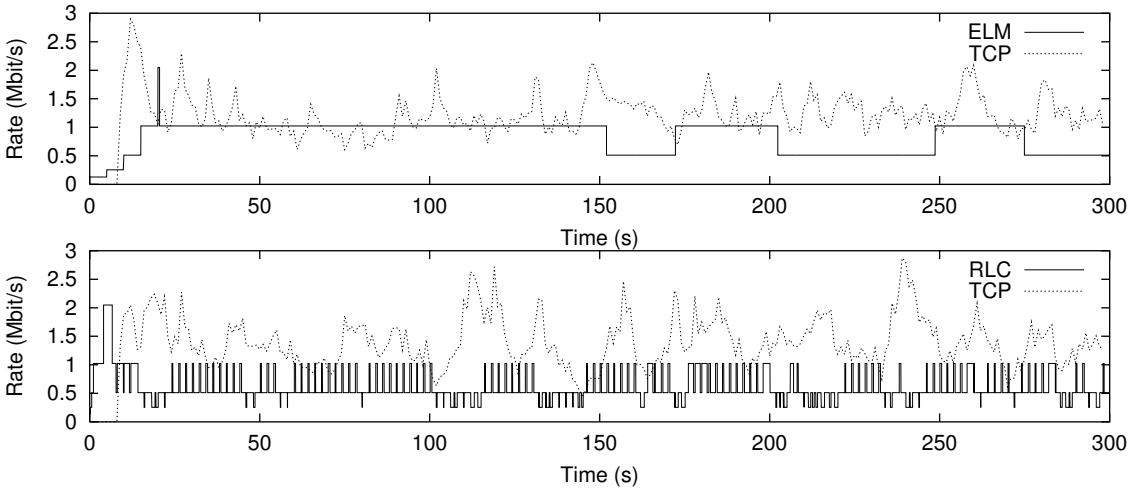


Figure 49: Simulation 5 at node R2. TCP burst from 200 to 220 seconds.

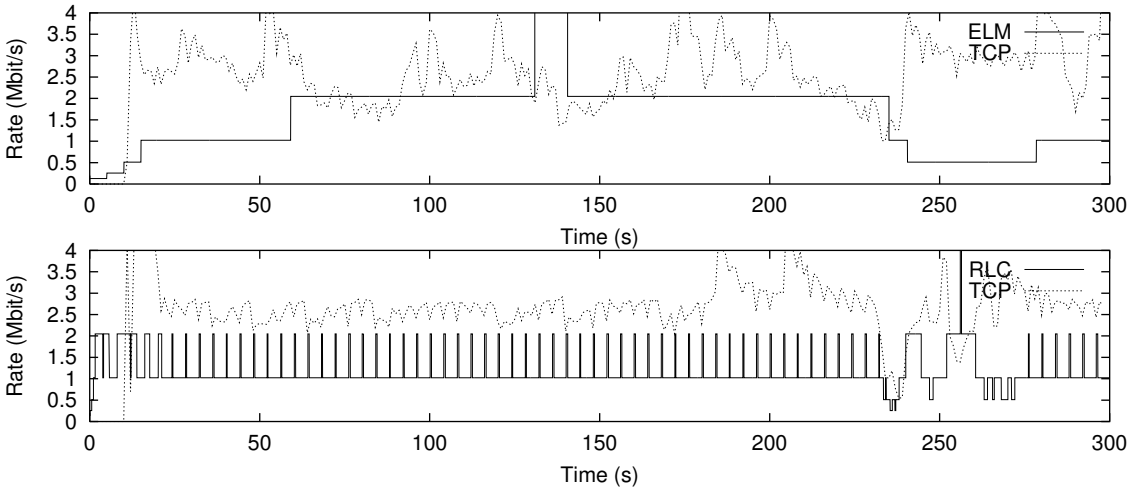


Figure 50: Simulation 5 at node R3. TCP burst from 230 to 240 seconds.

6.4 Summary

In this Section we have discussed several layer subscription management strategies that could benefit from equation-based rate estimation for receiver-driven multicast congestion control. Having a reliable estimate of congestion and TCP-compatible data rate, complex strategies may be develop to optimize congestion control to suite specific application needs.

We discussed three different strategies: Naïve strategy, Skeptical strategy, and Lazy strategy. Naïve strategy directly applies the calculated rate and is simple to implement. Skeptical strategy implements timers to postpone join and leave decisions and performs join or leave only when a timer expires. When the calculated rate oscillates, the skeptical strategy can prevent layer oscillations.

We presented in detail the Lazy strategy that attempts to avoid layer oscillations by postponing join and leave decisions. While the skeptical strategy has fixed timers to postpone decisions, the lazy strategy constantly adjusts the decision time. When the calculated rate changes significantly, the lazy strategy reacts to the change quickly. We described the mechanism that provide delayed joins and leaves when the unfairness in small, but reacts quickly to greater unfairness.

We implemented Adaptive Rate Control Framework (ARCF) in NS-2 simulator to simulate the Lazy strategy. We implemented the Lazy strategy in ARCF as a nearly complete multicast congestion control protocol called *Equation-based Layered Multicast* (ELM). Using the simulator implementation, we simulated some basic multicast scenarios and showed the performance of our protocol compared to RLC. We showed that ELM efficiently avoids oscillation, is TCP-compatible and claims the fair share in many scenarios. However, further research is required to analyze other laziness functions, optimal delay values as well as different strategies. Our choice of laziness functions and parameters showed sometimes slow reaction in presence of transient congestion.

7 Conclusion

In this thesis we have investigated the applicability of equation-based approach to receiver-driven layered multicast congestion control with very slowly adaptive, or fixed, layers. A layered multicast session consists of a sender (we restricted our work to single-source sessions) emitting data to a number of cumulative layers and a set of receivers subscribed to a subset of the layers. In an equation-based congestion control approach, loss-rate, round-trip time and, TCP retransmission timeout are estimated and fed in an equation to give a TCP-compatible sending rate.

We used TCP-Friendly Rate Control (TFRC) protocol as a basis for the investigation. We broke its closed-loop control into an open-loop control by forcing it to send at constant bit rate (CBR) independent of the estimated level of congestion and calculated rate. TFRC has been previously extended to multicast in two protocols: single-rate TCP-Friendly Multicast Congestion Control (TFMCC) and multi-rate Smooth Multicast Congestion Control (SMCC) which is a derivation of TFMCC. These protocols implicitly assume that the calculated rate is applicable in an open-loop scenario, i.e., that the calculated rate is independent of the sending rate.

We conducted a series of testbed experiments and simulations to study the calculated rate dependency on the sending rate. The testbed experiments indicated that the calculated rate depends on the sending rate and, in particular, the loss event rate depends on the sending rate. Motivated by the indication, we designed a set of simulation scenarios to study the loss event rate dependency on the sending rate.

Using the simulations we showed that the TFRC loss event rate algorithm indeed has a bias. We also showed that the bias originates from the way the algorithm ignores packet losses. TFRC estimates loss fraction by interpreting packet losses as loss events. A packet loss triggering a loss event follows a round-trip time,

called Loss Insensitive Period (LIP), during which packet losses are ignored. At high packet rates (multiple packets per round-trip time) and packet loss rates the algorithm measures a weaker congestion signal leading to a too high calculated rate.

We modified the loss event rate algorithm to measure a loss event impact. Instead of ignoring packet losses, our modification aggregates them to constitute a loss event impact. This way the modification imitates packet loss rate. We call our algorithm Loss Aggregation Period (LAP).

Using simulations we showed that LAP, compared to LIP, more accurately measures congestion under wide range of conditions and makes the loss estimate better applicable to open-loop control scenarios. Such scenarios include multicast as well as self-limited unicast where, e.g., a codec or the application specifies coarse grained data rates to choose from. We identified very low degree of statistical multiplexing and low sending rates as a particularly hard scenario for both, the original and, the modified algorithm.

We discussed possible layer subscription management strategies that could benefit from the modified loss event rate algorithm and improved accuracy in open-loop scenario. For further study, we implemented Adaptive Rate Control Framework (ARCF) for the NS-2 simulator. The framework was designed to enable easy implementation of different rate controlled unicast and multicast congestion control protocols. We implemented a multicast congestion control protocol we call Equation-based Layered Multicast (ELM) with two strategies: Naïve and Lazy strategies. Naïve strategy directly applies the calculated rate. Lazy strategy attempts to avoid layer changes when the discrimination is small.

Using simulation we showed that Lazy strategy outperforms Naïve strategy. We then conducted a set of multicast simulations and compared ELM (with Lazy strategy) to Receiver-Driven Layered Congestion Control (RLC). All of the simulation results show that ELM oscillates significantly less than RLC and achieves better fairness to TCP. Our implementation allows parameter tuning. However,

due to time constraints, we have used intuitive values without studying what values give optimal results.

Our multicast simulations showed good results. However, due to time and space constrains, we were forced to focus on a subset of required simulations. A significant set of multicast simulations should be performed to show consistent behavior of the protocol. The simulations also indicated weaknesses that need to be further investigated. For example, very low degree of statistical multiplexing is hard. What methods best detect environments of low and high degrees of statistical multiplexing and how to smoothly merge the operation under both environments should be investigated. A significant weakness of the algorithms originating from TFRC is that they operate on packets of fairly constant size. Recently this issue has been addressed, but it should be investigated thoroughly under open-loop control scenario.

We believe our work is an interesting and valuable contribution to the research community. The modification of the loss event rate algorithm is implicitly required by many multimedia applications due to the self-limited nature of many such applications. Also, the presented subscription management strategies offer a basis for development of application specific management strategies.

References

ALLY01 Arthuraliya, S., Li, V. H., Low, S. H. and Yin, Q., REM: Active queue management. *IEEE Network*, 15,3(2001), pages 48–53.

APS99 Allman, M., Paxson, V. and Stevens, W., TCP congestion control. Request for Comments (Standards Track) 2581, Internet Engineering Task Force (IETF), April 1999.

BB98 B. Braden, e. a., Recommendations on queue management and congestion avoidance in the internet. Request for Comments (Informa-

tional) 2309, Internet Engineering Task Force (IETF), April 1998.

BHH+00 Byers, J., Handley, M., Horn, G., Luby, M. and Vicizano, L., More thoughts on reference simulations for reliable multicast congestion control schemes. Technical Report, Digital Fountain, Inc., August 2000. URL `http://www.cs.bu.edu/fac/byers/pubs/mrefsims.ps`. Outcome of an informal meeting at Digital Fountain, Inc. [URL verified on November 2nd 2004].

CDK+02 Cain, B., Deering, S., Kouvelas, I., Fenner, B. and Thyagarajan, A., Internet group management protocol, version 3. Request for Comments (Standards Track) 3376, Internet Engineering Task Force (IETF), October 2002.

FF99 Floyd, S. and Fall, K., Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 7,4(1999), pages 458–472.

FGM+99 Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T., Hypertext transfer protocol. Request for Comments (Draft Standard) 2616, Internet Engineering Task Force (IETF), June 1999.

FHPW00a Floyd, S., Handley, M., Padhye, J. and Widmer, J., Equation-based congestion control for unicast applications: the extended version. Technical Report, International Computer Science Institute (ICSI), Berkeley, California, USA, March 2000. URL `http://www.aciri.org/tfrc/`. [URL verified on November 2nd 2004].

FHPW00b Floyd, S., Handley, M., Padhye, J. and Widmer, J., Equation-based congestion control for unicast applications. *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, August 2000, ACM Press, pages 43–56.

FJ93 Floyd, S. and Jacobson, V., Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1,4(1993), pages 397–413.

FML+03 Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T. and Diot, S., Packet-level traffic measurements from the sprint IP backbone. *IEEE Network*, 17,6(2003), pages 6–16.

HFPW03 Handley, M., Floyd, S., Padhye, J. and Widmer, J., TCP-friendly rate control (TFRC): Protocol specification. Request for Comments (Standards Track) 3168, Internet Engineering Task Force (IETF), January 2003.

Jac88 Jacobson, V., Congestion avoidance and control. *Symposium proceedings on Communications architectures and protocols*, volume 18, Stanford, California, United States, August 1988, ACM Press, pages 314–329.

Jai91 Jain, R., *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

JCH84 Jain, R. K., Chiu, D.-M. W. and Hawe, W. R., A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report DEC-TR-301, Digital Equipment Corporation, Eastern Research Lab, Hudson, MA, USA, September 1984.

KB03 Kwon, G.-I. and Byers, J., Smooth multirate multicast congestion control. *Proceedings of IEEE Infocom*, volume 2, San Francisco, California, USA, March 2003, IEEE, pages 1022–1032.

KHF04 Kohler, E., Handley, M. and Floyd, S., Datagram congestion control protocol (dccp). Internet draft, Internet Engineering Task Force (IETF), August 2004.

KR00 Kurose, J. F. and Ross, K. W., *Computer Networking*. Addison-Wesley, 2000.

LB00 Legout, A. and Biersack, E. W., Pathological behaviors for RLM and RLC. *Proceedings of NOSSDAV*, Chapel Hill, North Carolina, USA, June 2000, pages 164–172.

Low03 Low, S. H., A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions on Networking*, 11,4(2003), pages 525 – 536.

LPD02 Low, S., Paganini, F. and Doyle, J. C., Internet congestion control. *IEEE Control Systems Magazine*, 22,1(2002), pages 28–43.

MJV96 McCanne, S., Jacobson, V. and Vetterli, M., Receiver-driven layered multicast. *Proceedings of ACM SIGCOMM*, volume 26, New York, USA, August 1996, ACM Press, pages 117–130.

MMFR96 Mathis, M., Mahdavi, J., Floyd, S. and Romanow, A., TCP selective acknowledgment options. Request for Comments (Standards Track) 2018, Internet Engineering Task Force (IETF), October 1996.

MSM97 Mathis, M., Semke, J. and Mahdavi, J., The macroscopic behavior of the TCP congestion avoidance algorithm. *SIGCOMM Computer Commununication Review*, 27,3(1997), pages 67–82.

NB98 Nonnenmacher, J. and Biersack, E. W., Optimal multicast feedback. *Proceedings of IEEE Infocom*, volume 3, San Francisco, California, USA, March 1998, pages 964–971.

PFTK98 Padhye, J., Firoiu, V., Towsley, D. and Kurose, J., Modeling TCP throughput: a simple model and its empirical validation. *Proceedings of the ACM SIGCOMM*, Vancouver, British Columbia, Canada, September 1998, ACM Press, pages 303–314.

Pos80 Postel, J., User datagram protocol. Request for Comments 768, August 1980.

Pos81a Postel, J., Internet protocol. Request for Comments 791, September 1981.

Pos81b Postel, J., Transmission control protocol. Request for Comments 793, September 1981.

Ram00 Ramalho, M., Intra- and inter-domain multicast routing protocols: Survey & taxonomy. *IEEE Communications Surveys & Tutorials*, 3,1(2000). URL `http://www.comsoc.org/livepubs/surveys/public/1q00issue/ramalho.html`. Available in the Internet only [URL verified on November 2nd 2004].

RFB01 Ramakrishnan, K., Floyd, S. and Black, D., The addition of explicit congestion notification (ECN) to IP. Request for Comments (Standards Track) 3168, Internet Engineering Task Force (IETF), September 2001.

RSS03 Rimac, I., Schmitt, J. and Steinmetz, R., Observations on Equation-based Estimation of TCP-compatible Rate for Multi-rate Multicast Scenarios. *Proceedings of the International Workshop on Multimedia Interactive Protocols and Systems (MIPS 2003)*, Napoli, Italy, November 2003.

VCR98 Vicisano, L., Crowcroft, J. and Rizzo, L., TCP-like congestion control for layered multicast data transfer. *Proceedings of IEEE Infocom*, volume 3. IEEE, March 1998, pages 996–1003.

WBB04 Widmer, J., Boutremans, C. and Boudec, J.-Y. L., End-to-end congestion control for TCP-friendly flows with variable packet size. *SIGCOMM Computer Communication Review*, 34,2(2004), pages 137–151.

WH01 Widmer, J. and Handley, M., Extending equation-based congestion control to multicast applications. *Proceedings of ACM SIGCOMM*, San Diego, California, United States, August 2001, ACM Press, pages 275–285.