Equation-Based Congestion Control

Diploma Thesis

of

Jörg Widmer

from

Weinheim

Prof. Dr. Effelsberg

Practical Computer Science IV

Department of Mathematics and Computer Science

University of Mannheim

February 2000

Advisor: Mark Handley

Abstract

In this thesis, we introduce and analyze the *TCP-Friendly Rate Control Protocol* (TFRC), a rate-based end-to-end congestion control protocol. TFRC uses a model for steady state TCP throughput to limit the sending rate and assure fair behavior against competing flows. Instead of reacting to single congestion events (in the form of packet loss) like TCP, the TFRC protocol changes its sending rate in response to the loss rate, sampled over a certain amount of time. While TFRC achieves the same long-term throughput as a conformant TCP flow, its short-term sending rate is more stable. This makes the protocol suitable for traffic where sudden rate changes are undesirable (e.g. video or audio streams). Furthermore, rate-based congestion control is a promising basis for large scale multicast transport protocols. Since no router support is necessary, the protocol can be readily deployed in today's Internet.

Acknowledgements

This work drew inspiration and support from a number of people to whom I wish to express my gratitude.

First of all, I would like to thank Prof. Dr. Effelsberg. Without his support, this thesis would not have been possible.

I would like to thank Mark Handley, Sally Floyd, and Jitendra Padhye of the ACIRI group for their insightful comments and invaluable support.

I am indebted to Rajeev Koodli (Nokia) and Barry Stein (UCL). Many of the Internet experiments would not have been possible without their technical support.

Furthermore, I would like to thank Joachim Beer and the other members of the Network Services and Applications group of the ICSI for their much appreciated ideas and feedback. Special thanks also to Ky-Van Lee and Claudia Osmann for their detailed comments on the work.

Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledgement
AIAD	Additive Increase, Additive Decrease
AIMD	Additive Increase, Multiplicative Decrease
DSL	Digital Subscriber Line
ECN	Explicit Congestion Notification
EWMA	Exponentially Weighted Moving Average
FEC	Forward Error Correction
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP[v4|v6]	Internet Protocol (version 4 / version 6)
ISM	Inter-packet Space Modulation
LAN	Local Area Network
MIAD	Multiplicative Increase, Additive Decrease
MIMD	Multiplicative Increase, Multiplicative Decrease
NACK	Negative Acknowledgement
QoS	Quality of Service
RED	Random Early Detection
RTCP	Real-Time Control Protocol
RTO	Retransmission Timeout Value
RTP	Real-Time Transport Protocol
RTT	Round-Trip Time
SACK	Selective Acknowledgement
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TDACK	Triple Duplicate ACK
TFRC	TCP-Friendly Rate Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network

Chapter 1

Introduction

The first wide area networks (WANs) were circuit-switched telephone networks. In these telephone networks, the number of simultaneous users is limited, but when a connection is established, service is guaranteed for the duration of the connection for each user. This guarantee causes resources to be blocked once they have been claimed by a user. Full utilization of network resources is difficult to attain in networks that feature these service guarantees.

For this reason, today's internetworks are mostly best effort networks. The number of simultaneous users is unlimited, but service is not guaranteed. In these so-called store-and-forward networks, data packets are sent from the source to intermediate routers. A router stores the packets until they can be transferred to the next router. This continues in succesion until the packets ultimately reach their destination.

1.1 Congestion

When the demand for bandwidth exceeds available network resources, network paths can get congested. If the short-term packet arrival rate is higher than the maximum packet processing rate of a router or the outgoing link bandwidth of a router, packets are temporarily stored in a buffer. A router can sustain high packet arrival rates for short periods of time, where the duration of the period depends on the buffer size. However, when the buffer is full the router has to discard packets. Connections going over this router will therefore experience losses. For reliable data transfer, lost packets need to be retransmitted further increasing the load on the network. Under unfavorable conditions, network congestion can be sustained by retransmissions even when the original cause of the congestion is no longer present.

Several causes for congestion can be identified:

- Long distance, high-speed connections increase the total amount of data in the network. When the amount of data exceeds the capacity of the network, the overflow is stored in buffers. The higher the buffer utilization in the network, the higher the probability of packet loss.
- Even when the average load on the network is not critical, bursts of network traffic can cause temporary congestion.
- Unfavourable topologies and mismatching link speeds also contribute to congestion, for example slow

routers that connect several traffic intense hosts to the Internet.

- Some protocols do not sufficiently decrease the sending rate when congestion occurs. Applications using such protocols pose an increasing problem as they become more widespread. The Internet relies on the cooperation of communication end-points to prevent a congestion collapse.
- The fast growth of the Internet will be a problem if the supporting infrastructure is unable to keep up with the increase in network traffic. Over the past twenty years, the number of users and the speed of their connections to the Internet has increased by several orders of magnitude. In addition, the usage of the Internet has grown with an increasing number of services being replaced or supplemented with their e-counterparts (e.g. e-mail, e-business, etc.). Existing applications are upgraded to support higher bandwidths as they become available. New high-volume applications that were previously possible only in local area networks have been moved to the Internet.

Congestion is existent on various timescales, ranging from transient congestion to more long-term congestion persisting for months or even years. Transient congestion might exist for such a short period of time that it dissipates before it is possible to react to the congestion. Long-term congestion is usually caused by an inadequate network infrastructure and will tend to last as long as the infrastructure remains unchanged. The focus of this thesis is on congestion control on an intermediate timescale not longer than the duration of the connection (i.e. from fractions of a second up to hours). Congestion on smaller and larger timescales cannot be influenced by means of end-to-end congestion control protocols.

1.2 Congestion control

Packet loss at routers due to congestion can be problematic. Transport protocols that offer reliable data transfer need to retransmit dropped packets causing additional network traffic. Congestion management is imperative in order to allow the network to recover from congestion and operate in a state of low delay and high goodput[1].

The Transmission Control Protocol (TCP) is the most commonly used transport protocol in the Internet. In the first years of the deployment of TCP, it had only a very rudimentary congestion control mechanism that was not sufficient in preventing congestion in intermediate routers. As a result, the Internet experienced several severe congestion collapses in the mid 80s. During a congestion collapse, only a very small fraction of the existing bandwidth is utilized by traffic that ultimately reaches the receiver. Most of the traffic consists of packets that are discarded on their path through the network.

The cause for the congestion collapses was examined for example by Jacobsen in his groundbreaking work [Jac88]. He proposed a modified congestion control mechanism for TCP. It is remarkable that TCP is still in use today despite the fact that it was developed 20 years ago. TCP's success is due mainly to its now robust congestion control mechanism. This mechanism causes TCP to reduce its sending rate when congestion is encountered along the network path, as evidenced by dropped packets.

TCP is well suited for most of the tasks for which it is used today, but there are applications that have transport protocol requirements that TCP cannot fulfill. Not all applications require reliable data transfer. Some applications may use alternative mechanisms to recover from transmission errors as for example

[1] Goodput is defined as "the bandwidth delivered to the receiver, excluding duplicate packets" [FF99].

Forward Error Correction (FEC). For other applications, retransmitted data may be outdated by the time it is received so that retransmissions would be useless. Furthermore, TCP reduces its sending rate by a large amount in response to a single lost packet. While this is a good way to avoid or reduce congestion in the network, some applications may not be able to cope with such an abrupt rate reduction (e.g. audio and video transmissions).

Typically, applications use the User Datagram Protocol (UDP) when TCP is not suitable for the task. UDP offers an unreliable data transfer service (i.e. it does not try to recover lost packets). UDP also does not back off in times of congestion the way TCP does but is completely congestion-unaware. Not reducing the rate in response to congestion can cause the network to waste bandwidth and is unfair to competing protocols that are congestion-aware [FF99]. Some applications using FEC[2] will even increase the redundancy level and thus their data rate in an environment with high loss rates. This further increases the load on the already congested network. While FEC improves the quality of service of the application, FEC can severely harm other flows or the network when it is not combined with a congestion control mechanism.

Thus far, this has not been a serious problem because TCP makes up about 95% of all Internet traffic [CMT98]. Furthermore, the Internet connections of the majority of end-users were very slow, for example over a 28.8KBit/s modem line. With the advent of new technologies such as cable modems or Digital Subscriber Lines (DSL) the end-user now has a link bandwidth in the range of MBits/s. Congestion control consequently becomes increasingly important. A higher bandwidth permits new applications such as video conferencing tools or online gaming. These applications increase the percentage of non-TCP-based traffic in the Internet because they often cannot use the TCP protocol. In addition, all multicast traffic [Dee91] is based on UDP and thus has no back off mechanism.

While all the previously mentioned reasons to use congestion control benefit the network, there are advantages for the application as well. For example, in large scale multiplayer online games and telecooperation based simulations, a low delay and high responsiveness is of major importance to maintain a feeling of "realism". When participants send data in disregard of the congestion level, available buffers in the network are likely to fill up. This renders the application unusable because of the increased buffer delay[3]. With congestion control, there is less delay in the network which is more important than a high data rate for these types of applications.

1.3 Design space for a congestion control protocol

The following characteristics are desirable for an end-to-end congestion control protocol:

1. Stability: a sending rate that remains fairly stable even with an increased level of noise in the network
2. Wide adaptive range: the ability to sustain performance over a wide range of network conditions and cope with heterogeneity in the network.
3. Fairness: the protocol should compete fairly with other flows, in particular TCP flows.
4. Performance: no performance penalty resulting from low computational overhead and little control traffic; no underutilization of resources caused by the congestion control scheme.
5. Responsiveness: fast response to permanent changes in network conditions.

[2] FEC offers additional reliability at the expense of an increase in the amount of data to be sent.

[3] In the network experiments presented in chapter 4, applications experienced buffer delays of up to several seconds.

Not all of these characteristics can be achieved at the same time; a tradeoff exists between the responsiveness and the stability of the protocol. When the sending rate is more stable and less sensitive to noise, it will be less responsive to changes in the network. A similar tradeoff exists between a wide adaptive range and protocol responsiveness. On the other hand, fairness and good performance should be consistently achieved by the protocol.

Several possible forms of congestion control exist. First, in the case of *sender driven* congestion control, the sender changes its rate in response to the current network conditions. Second, *receiver driven* congestion control shifts the responsibility to the receiver. Receivers join or quit a session (or subsession in a *layered* congestion control scheme) depending on the current congestion level. Third, in *router driven* congestion control, routers identify flows that use more than their fair share of bandwidth in order to preferentially discard packets of such flows [FF99]. An optimal solution would consist of either a sender or receiver driven scheme supplemented by router driven congestion control.

Applications are better able to adjust their sending rate[4] than routers which can only discard packets. Presently there are no incentives for applications to do so. Applications that utilize more than their share of bandwidth are "rewarded" with a higher throughput. This comes at the expense of reduced throughput for competing flows and a higher congestion level in the network. In order to install such an incentive, routers must penalize flows that fail to employ an adequate congestion control mechanism by overproportionally reducing their throughput during congestion. While this is the most promising approach, the Internet currently lacks adequate congestion control enforcement by the routers. Since the router infrastructure in the Internet is slow to change, there is no alternative to end-to-end congestion control without router support for the time being.

This thesis examines sender driven congestion control in the form of the TCP-Friendly Rate Control Protocol (TFRC). Similar to the TCP protocol, the TFRC protocol interprets packet loss as an indication of congestion and reduces its sending rate accordingly. TFRC's average sending rate is similar to that of TCP. Since TFRC increases and reduces its sending rate over a longer period of time, its short-term rate varies less compared to that of TCP. The stability of the short-term rate makes TFRC well suited for rate-adaptive applications requiring gradual rate changes.

It is necessary to separate congestion control from application-level reliability since not all applications have the same error recovery requirements. While TFRC is designed to be an unreliable transport protocol, reliability can easily be implemented on the application layer according to the needs of the application.

To make the protocol easy to deploy, it is important to be as independent of scales as possible and to use few configuration parameters. Scaling parameters are difficult to gather automatically while manual configuration is prone to human errors.

The protocol is implemented at the communication end-points and does not require additional information from routers, for example buffer utilization or link bandwidth. The primary design goal is to support applications that are unable to use TCP due to different application requirements. TFRC is intended to be an alternative and not a substitute for TCP.

[4] for example by spatial scaling of a video stream

1.4 Outline of the thesis

The second chapter describes the TCP protocol, in particular features of TCP that affect coexistence with the TFRC protocol (e.g. the congestion control and slowstart mechanism). Models for TCP throughput that can be used in rate-based congestion control are delineated. In the third chapter, the TFRC protocol for unicast congestion control is introduced; details about characteristics of the protocol are given. TFRC's congestion control mechanism is based on a model of TCP. How the model is integrated with the TFRC protocol is discussed. In chapter 4, we present results obtained from simulations and tests with *Dummynet* [Riz97] and the public Internet. Specific metrics are defined in order to evaluate TFRC performance and fairness to TCP. Related work is addressed in chapter 5. In the final chapter, results are summarized and issues not covered in this thesis are reviewed. In particular, we discuss problems that have to be solved in order to deploy the protocol in multicast environments.

Chapter 2

The Transmission Control Protocol

Most network protocols consist of several independent layers to simplify protocol design and reduce complexity. Each layer is responsible for a specific task of the communication. In particular, lower network layers offer services to higher layers. The TCP/IP protocol stack consists of four layers, namely the *link* layer, the *network* layer, the *transport* layer, and the *application* layer. The link layer includes low level components such as the network device driver and the necessary hardware. The network layer is responsible for routing and delivery of packets in the network. In case of the TCP/IP protocol suite, it consists mainly of the Internet Protocol (IP) as specified in [Pos81a].[1] IP offers an unreliable connectionless datagram delivery service. The transport layer provides reliable or unreliable end-to-end data transfer. Most commonly known examples for transport protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) as described in [Pos81b] and [Pos80] respectively. The fourth layer is the application layer. It comprises applications such as telnet, FTP, or SMTP that use the network functionality provided by the lower layers. Usually, the lower three layers are part of the kernel of the host operating system, while the application layer is a user process [Ste98].

The main concern of this thesis is the transport layer. In the following sections we present details of the TCP protocol. The understanding of these features is important as they influence the design of the TFRC protocol to a large extent.

2.1 Transmission Control Protocol

TCP is a connection-oriented protocol. Before data can be transmitted between two network end-points, a connection has to be established. The connection is *full-duplex* (i.e. data can be exchanged between the two hosts simultaneously in both directions). TCP offers reliable data transfer. Thus, all transmitted data will ultimately reach the receiver. Since the IP layer does not guarantee packet delivery, TCP has to retransmit lost packets. The receiver acknowledges each packet it receives. Packets that are not acknowledged within a certain time span are retransmitted by the sender. When a duplicate packet arrives at the receiver, either because it was duplicated by one of the lower network layers or erroneously retransmitted, it is discarded.

[1] Other components are the Internet Control Message Protocol (ICMP) and the Internet Group Management Protocol (IGMP).

Packets that are received out of order are reordered to deliver exactly the byte stream to the receiver that was sent by the sender. A *sequence number* that corresponds to the offset from the beginning of the sent byte stream is assigned to each packet. The receiver returns acknowledgements (ACKs) for received packets to the sender.[2] These ACKs carry the sequence number of the next expected packet (i.e. the number of bytes that were received in order up to present incremented by one). When both communication end-points have data to send TCP, can piggyback ACKs (i.e. ACKs are sent in combination with a data packet to reduce the amount of control traffic).

2.1.1 Flow control

A mechanism called *flow control* prevents, that a fast sender overflows the receive buffer of a slower receiver. The sender is only allowed to send as much data as the receiver can cope with. A *sliding window* at the sender and at the receiver limits the number of outstanding (unacknowledged) packets in the network. Since sending data reduces the window size, the sender can only send packets until the window limit is reached. Then it has to wait until acknowledgements increase the window size again to be able to send further packets.

2.1.2 Self-clocking characteristics of TCP

The temporal spacing of the data packets depends on the link capacitity (i.e. the packet frequency increases with the available bandwidth). In equilibrium, TCP only allows sending of a packet when another packet is acknowledged. The ACKs from the receiver arrive at the TCP sender at the same pace at which the data packets leave the sender. The algorithm is self-clocking and provides a very elegant way to limit the number of outstanding packets in the network.

The time it takes a packet to travel from sender to receiver and the time it takes the ACK to travel back to the sender is called the round-trip time (RTT). The sender has to be able to continually send data at the link bandwidth during this time to be able to fully utilize the available bandwidth. Thus, the size of the sliding window should be equivalent to the link capacity or the *bandwidth delay product*. The bandwidth delay product s_{bdp} is defined as round-trip time t_{RTT} times the link capacity b_{lnk}.

$$s_{bdp} = b_{lnk} \cdot t_{RTT} \tag{2.1}$$

2.1.3 Slowstart

The window policy allows a sender to send a whole window of data at the maximum possible rate. Sending a burst of packets until the advertised window is used up or acknowledgements increase the window size again is not a problem as long as sender and receiver are on the same LAN. However, connections to hosts on other networks will usually go through (several) intermittent routers with limited resources. Inserting a large number of packets at a high rate can cause severe congestion and drastically reduce the goodput of the connection.

Jacobsen [Jac88] presents a solution to this problem in form of a *slowstart* algorithm. This mechanism requires a second window called the congestion window *cwnd*. The size of this window is initialized to

[2] Many TCP implementations send a cumulative ACK every other packet to reduce the amount of control traffic [Ste94].

one segment[3] when the communicating hosts are not located on the same network. The sender has to use the minimum of both the congestion window and the receiver window as an upper bound for the number of outstanding packets. When a new connection is established, the sender can send one segment and then has to wait for the corresponding acknowledgement. Each received ACK increases the congestion window by one segment. Since also the acknowledged packet can be discarded from the window, the sender can now send two packets. Again, when the acknowledgements arrive, the sender can discard two packets from the congestion window and in addition increase the window size by two segments, resulting in a total *cwnd* size of four, etc. The increase of the congestion window is exponential. Its size doubles every RTT until an equilibrium is reached and the window advertised is fully utilized.

2.1.4 Loss detection

In an ACK-based protocol, the sender is responsible for the detection of packet losses. Lost packets are revealed by flaws in the order of acknowledged sequence numbers due to missing ACKs. The TCP sender can identify lost packets in two ways, via a timeout or a triple duplicate ACK.

Timeouts

TCP waits a certain time interval for an acknowledgement to arrive. If the ACK is not received within this time interval, the corresponding packet is assumed lost and consequently retransmitted. The retransmission timeout value (RTO) is crucial to the protocol performance. When it is too large, the protocol performs poorly by spending too much time waiting for the arrival of lost packets. When the interval is too small, packets might be retransmitted, although the original data packet was not lost and the ACK was still on its way. This unnecessarily increases the load on the network. Obviously, the timeout value should be closely related to the RTT. The smoothed RTT is computed as the exponential moving average of RTT samples.

$$t_{RTT} = \frac{1}{8}\, t_{RTT\,sample} + \frac{7}{8}\, t_{RTT} \tag{2.2}$$

RTT samples can be deduced from the time interval between the sending of a data packet and the reception of the corresponding ACK.

Originally, the RTO was merely a multiple of the RTT. Connections experiencing a high RTT variance performed poorly because of too many unnecessary retransmissions. Performance was considerably improved by taking the variance t_{RTTvar} into account to compute a more accurate timeout value t_{RTO} [Jac90].

$$t_{RTTvar} = \frac{1}{4}\, |t_{RTT} - t_{RTT\,sample}| + \frac{3}{4}\, t_{RTTvar} \tag{2.3}$$

$$t_{RTO} = t_{RTT} + 4\, t_{RTTvar} \tag{2.4}$$

The computation is simplified by using the mean deviation of the RTT samples as an approximation of the standard deviation, which would require a square root. To be able to use integer arithmetic, decay factors are powers of two.

The RTO has a large impact on the average TCP sending rate when the packet drop rate in the network is high. In case the congestion that caused a timeout is persistent and even retransmitted packets are not

[3] One segment corresponds to one data packet.

acknowledged within the timeout interval, TCP backs off exponentially by doubling the timeout interval every time it tries to retransmit the packet.

Triple duplicate ACKs (TDACKs)

Waiting for the retransmission timer to expire causes a considerable delay. To speed up retransmission of lost packets when data is still exchanged between sender and receiver, a *fast retransmit/fast recovery* mechanism was added to TCP. Whenever a packet is received out of order, an ACK is sent with the sequence number that was actually expected (duplicate ACK) to inform the sender. The sender does not know if the duplicate ACK was caused by a lost packet or simple packet reordering. However, when three consecutive duplicate ACKs are received, the sender assumes that the corresponding packet was lost and retransmits it without waiting for the retransmission timer to expire. Details of this mechanism are given for example in [Ste94].

2.1.5 TCP congestion control

As mentioned above, the number of outstanding packets is constant in steady state. This prevents TCP from claiming additional bandwidth that might have become available after TCP performed slowstart. To detect additional bandwidth, TCP increases its sending rate also after slowstart is finished. The increase during normal operation has to be much smaller than the doubling of the sending rate as in slowstart mode. Each time TCP receives an ACK, the congestion window size $cwnd$ is increased by $1/cwnd$. Since roughly $cwnd$ packets are acknowledged during one RTT, the overall increase of $cwnd$ is one segment per RTT (*additive increase*).

When the TCP connection experiences congestion indicated by a timeout or a triple duplicate ACK, it backs off by decreasing its sending rate. This allows the network to recover from its congested state. In case of a triple duplicate ACK, TCP reduces its congestion window and thus the sending rate by half (exponential or *multiplicative decrease*). Since the flow of packets between sender and receiver is uninterrupted, the sender then continues with normal congestion control by slowly increasing $cwnd$ again. A timeout on the other hand indicates that no more packets were received by the receiving host or all the acknowledgements were lost on their way to the sender. This is a hint for severe congestion. In this case, the sender sets its congestion window to one and performs slowstart again to determine an appropriate sending rate. Most TCP implementations reduce their sending rate in response to a packet loss only once per window. Further losses in the same window are ignored.

The combined mechanism of additive increase and multiplicative decrease is termed AIMD. It has very good stability properties compared to other congestion control mechanisms such as AIAD, MIMD, and MIAD. A comparison of different increase/decrease schemes is presented in [CJ89].

2.1.6 Different flavors of TCP

Since the first TCP implementations, TCP has over time been improved in several ways. Today, many different versions of TCP are in use. This section gives a brief overview over some of the more important

versions of TCP.[4]

Early implementations of TCP: Use of a go-back-n retransmission scheme with cumulative acknowledgements and a retransmission timer for the recovery of lost packets. The timeout value is a multiple of the RTT.

Tahoe: Additional features such as slowstart, congestion avoidance, fast retransmit, and a refined timeout value calculation as described in [Jac88].

Reno: Fast retransmission is modified to include fast recovery[5] and no slowstart is performed after a triple duplicate ACK. This results in increased performance in case of single packet drops.

New Reno: Small modifications to TCP Reno to improve performance in case of multiple packet drops in a window.

Sack: Selective acknowledgements instead of go-back-n to further improve performance under heavy congestion with multiple packet losses per window, as proposed in [MMFR96].

2.1.7 TCP performance

The granularity of the TCP timers can vary from 10ms to 500ms, depending on the TCP implementation of the host systems [Ste94]. These implementation specific details can result in a considerable throughput discrepancy of different TCP variants under the same network conditions. [AP99] investigates the effect of different clock granularities and various RTT and timeout computation methods.

To avoid congestion on a network path, it would be sufficient for TCP to adapt to the available bandwidth at the link with the heaviest congestion and only react to packets loss that occurs at that specific router (local fairness). Since TCP reacts to all loss events regardless of where they occur, TCP behaves more conservative than necessary when crossing multiple congested gateways [WC98]. On the other hand, more routers benefit from a bandwidth reduction of a flow that uses resources on several congested gateways which is a good justification for TCP's conservative behavior.

2.2 Models for TCP throughput

TCP throughput is influenced by many parameters. It is too complex to be modeled in all of its aspects and for all possible network conditions. However, if simplifying assumptions are made, models can provide a reasonable estimate of TCP's steady state throughput. They do not to accurately estimate short-term throughput during a single RTT, but average long-term throughput over timescales of many RTTs.

A model for TCP defines a function that relates throughput b, round-trip time t_{RTT}, the average number of window reduction events per packet l, the segment size s and (for the complex model) the timeout value t_{RTO}.

$$b = f(t_{RTT}, l, s, t_{RTO}) \tag{2.5}$$

[4] For a more thorough discussion see [FF96].

[5] For reasons of brevity, fast retransmit and fast recovery are not discussed here, but the reader is refered to [Pax97a] and [Ste94].

2.2.1 Simple model

A simple model for TCP throughput was first presented in [MF97] and later more thoroughly studied in [OKM96] and [MSMO97]. The throughput of a TCP connection is given as:

$$b = c\,\frac{s}{t_{RTT}\,\sqrt{l}} \tag{2.6}$$

The constant c is in the range of 0.87 to 1.31, depending on the utilization of delayed ACKs and on the assumption of periodic or random loss events. The model is not tailored to a specific TCP variant. In the range of network conditions where this model is accurate, the different TCP variants show very similar behavior.

The model makes the following assumptions:

- TCP experiences window reduction events only because of triple duplicate ACKs, not because of timeouts. This is a reasonable assumption when the loss rate is low. At higher loss rates, the congestion control mechanism is more and more dominated by timeout events and the model overestimates TCP throughput.
- TCP experiences at most one window reduction event per RTT.
- Loss events have a fixed probability and are mutually independent.
- RTT and loss rate are independent from the sending rate. This holds true for connections with a high level of statistical multiplexing where the behavior of one flow has only little impact on the state of the network.

An equation for TCP throughput can be derived by estimating the average size of the TCP congestion window. Assume periodic loss events and a congestion window of size W. A loss event causes a reduction of the window size to $1/2\,W$. No other loss events occur for the next $1/l$ packets because of the periodicity assumption. During this time, the congestion window size increases by one segment per RTT. After $W/2$ RTTs, the congestion window reaches its original size W again. Figure 2.1 shows the size of $cwnd$ throughout several increase/decrease cycles. The number of packets delivered in one cycle (see shaded area in figure 2.1) is

$$\left(1+\frac{1}{2}\right)\cdot\left(\frac{W}{2}\right)^2 = \frac{3\,W^2}{8}$$

Since there occurs exactly one loss per cycle, loss rate and window size are related as follows:

$$l = \frac{8}{3W^2} \qquad \text{or} \qquad W = \sqrt{\frac{8}{3l}}$$

The bandwidth b can be computed as the amount of data transmitted in a cycle over the duration of the cycle.

$$b = \frac{s\cdot\frac{3}{8}\,W^2}{\frac{1}{2}\,W\cdot t_{RTT}} = \frac{s/l}{\frac{1}{2}\sqrt{\frac{8}{3l}}\cdot t_{RTT}} = \sqrt{\frac{3}{2}}\,\frac{s}{t_{RTT}\,\sqrt{l}}$$

The model that corresponds to this method of deriving TCP throughput is presented in [MSMO97]. The derivation of the other models [MF97, OKM96] leads to a slightly different constant c, but the relationship of throughput, loss rate and RTT coincides with this model.

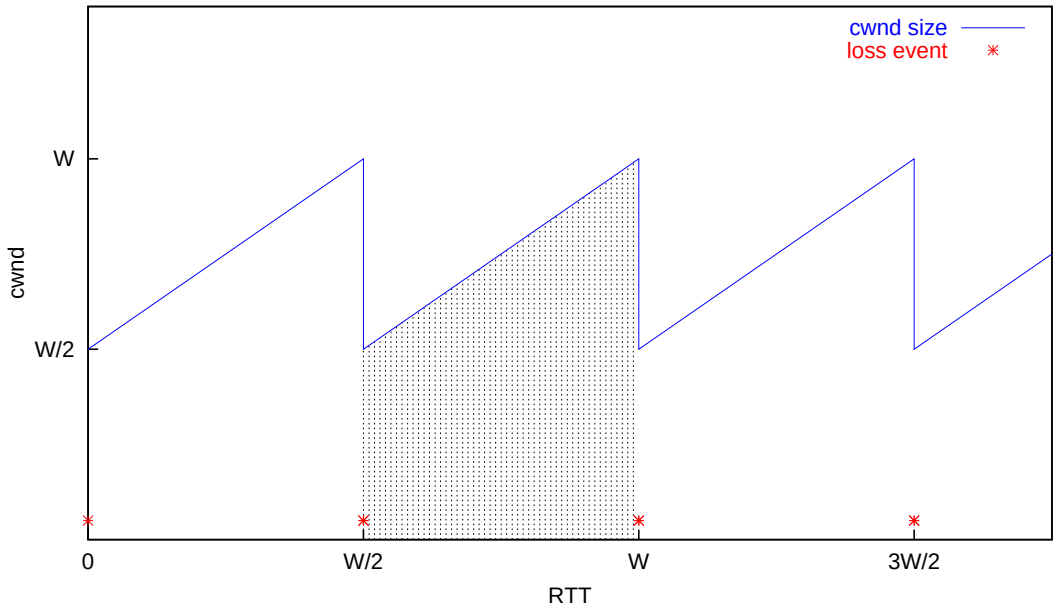


Figure 2.1: Congestion window size under periodic loss

2.2.2 Complex model

The simple model fails to accurately estimate TCP throughput under high loss rates. The more complex model presented in [PFTK98] and [PFT99] can be applied for a broader range of network conditions. It takes window reduction events caused by timeouts into account.

Obviously, the "no timeout" assumption from the simple model is dispensable in the complex model. The assumption of a maximum of one window reduction event per congestion window (or RTT) is changed to the assumption that when one packet is lost in a window all subsequent packets of the window are lost as well. This complies with general packet loss patterns experienced with drop tail routers. Furthermore, the model requires a refined concept of loss rate. The *loss event rate* is defined as the probability that a packet is lost, given that it is the first packet in a cycle or that *all* the previous packets in the cycle were not lost. This is closely related to TCP's behavior of reacting to at most one window reduction event per window.

The steady state throughput is analyzed using Markov processes. An approximation of the full model is given by

$$b(l) = \frac{s}{\underbrace{t_{RTT}\sqrt{\frac{2\,bl}{3}}}_{\text{TDACK}} + \underbrace{t_{RTO}\sqrt{\frac{3\,bl}{8}}\,l\,(1+32\,l^2)}_{\text{timeout}}} \tag{2.7}$$

where s is packet size, l is the loss event rate as defined above, b is number of packets acknowledged by each ACK, t_{RTT} is the RTT, and t_{RTO} is the TCP retransmission timeout value. The first term in the denominator accounts for TDACKs and resembles the denominator in equation 2.6. The second term models the impact

of TCP timeouts on the average sending rate. At high loss rates, this term dominates the equation.

Real-world experiments as presented in [PFTK98] show that this model renders TCP throughput far more accurately than the simple model. The model is adjusted to TCP with selective acknowledgements (Sack TCP). Sack TCP outperforms other TCP variants at high loss rates. Thus, the model may overestimate the throughput of some TCP implementations.

2.2.3 Comparison of the models

Figure 2.2 shows the TCP throughput calculated by the two models under various round-trip times and loss rates. The complex model and the simple model result in roughly similar throughput at low loss rates, while the throughput of the complex model decreases faster at higher loss event rates. In a highly congested network environment, nearly all of the TCP congestion window reductions are caused by timeouts. Thus, the complex model is far more accurate under such conditions.

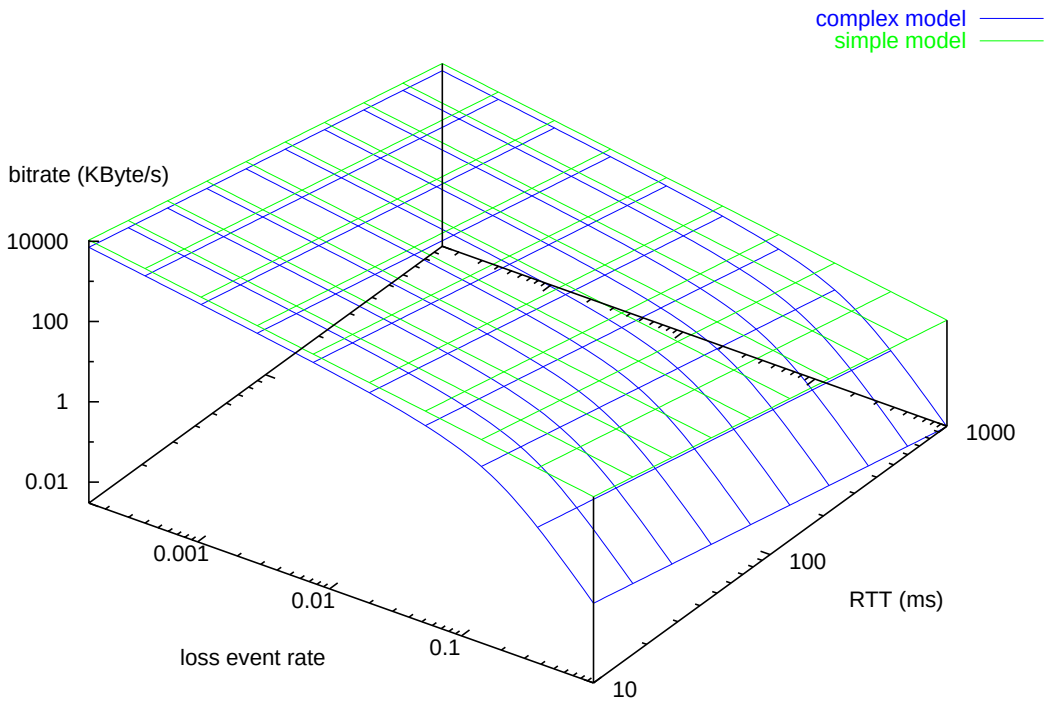


Figure 2.2: Throughput estimate of TCP the models under various network conditions

Note that applying these models is not without complications [BG99] and further research in modeling TCP is necessary. Yet, a more thorough discussion of the models is beyond the scope of this thesis.

2.3 TCP-friendliness

Non-TCP flows are considered *TCP-friendly* or *TCP-compatible* when "their long-term throughput does not exceed the throughput of a conformant TCP connection under the same conditions" ([FF99], [BCC+98]). As discussed before, no *typical* version of TCP exists. Different variants of TCP can achieve throughputs that vary considerably.

Also the loss patterns themselves influence TCP throughput. Some TCP implementations respond in a different manner to the same loss patterns. Flows may experience different packet loss patterns. This occurs especially when flows have different packet rates and results in reduced intra-protocol fairness. A TCP connection with one loss every other window achieves a different throughput than a connection with two losses every four windows although the average loss rate is the same.

Thus, a TCP-friendly flow should have a sending rate that is close to TCP throughput although it is not necessary to exactly follow the throughput of a specific TCP implementation. In any case, the sending rate has to be in the same order of magnitude as the corresponding TCP throughput.

Chapter 3

The TCP-Friendly Rate Control Protocol

Congestion control is of major importance for networks to allow operation in a stable range of network conditions. TCP's congestion control mechanism has proved to result in a stable network state. However, window-based mechanisms are not the only way to perform congestion control and they are not suitable for every application. Directly adjusting the sending rate to the level of congestion in the network is a viable alternative to a congestion window. In particular, rate-based congestion control is useful for applications that are inherently rate-based such as streaming multimedia. It is easy to modify the sending rate to adhere to application constraints such as timing constraints or delay jitter. Reliability and congestion control are decoupled in this rate-based scheme. This provides more flexibility than TCP's window mechanism that combines reliability and congestion control.

The fair sending rate is determined by the conditions on the network links such as the total link bandwidth, the amount of cross-traffic, the number of competing flows, and router internals such as queuing policy, buffer size, and buffer utilization. Since the TFRC protocol is intended to be employed in the communication end-points, it has no access to these informations. What can be measured are second order parameters such as round-trip time and loss rate. Feedback about these parameters can be used to infer the congestion state of the network and determine an appropriate sending rate.

3.1 Functioning of the protocol

The TFRC protocol consists of the following main components:

- a model to determine the (TCP-friendly) sending rate,
- methods to measure the necessary parameters,
- a feedback mechanism to return this information to the sender, and
- an increase/decrease policy to adjust the current sending rate.

While the sender transmits packets at a certain bitrate, the receiver keeps track of important network characteristics. This information is then used by the receiver to determine if the sending rate is appropriate or should be increased or decreased. The expected bitrate is communicated back to the sender (along with other information) in so called receiver reports. The sender reacts to the reports according to the increase/decrease

policy. Figure 3.1 gives an overview of the protocol components and shows how the components interact.

Receiver reports are sent once per RTT since this is the delay of the control system. It takes half an RTT until the receiver experiences the changes in the sending rate and another half RTT until the sender receives a report reflecting the altered network conditions caused by the change in the sending rate. When the sending rate is less than one packet per RTT, a report is only sent when at least one packet was received since the last report. This reduces the amount of control traffic.

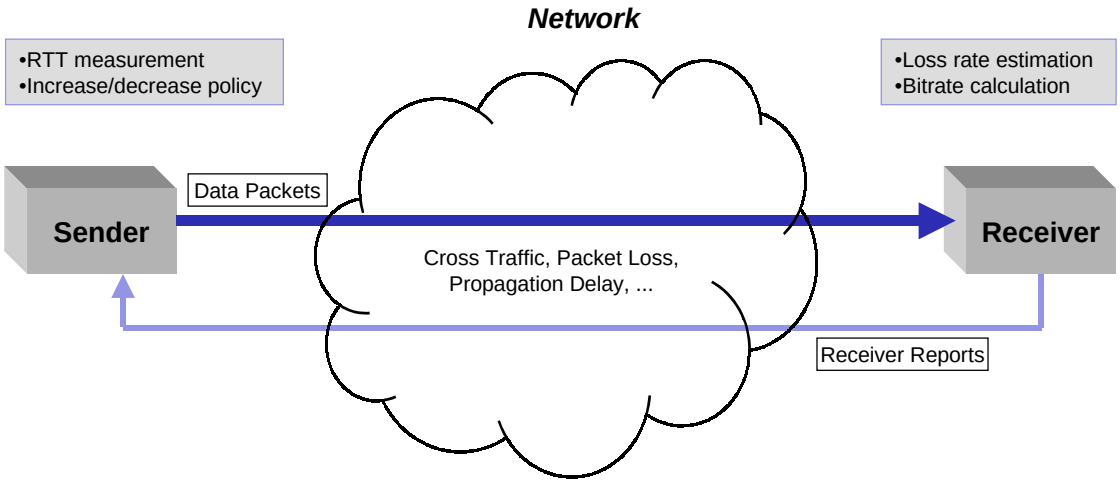


Figure 3.1: Control system

3.2 Application of the TCP model

To ensure that the TFRC protocol competes fairly with other protocols, especially TCP, over a wide range of network conditions, we use the more accurate model for TCP throughput as presented in chapter 2.2.2. Utilization of an equation to control the sending rate is a very flexible way of congestion management. This makes it easy to comply with constraints (e.g. delay, delay jitter, minimum bandwidth, etc.) imposed by the application.

Both the simple and the complex equation can only be applied when loss rate and RTT are greater than zero. The RTT will always meet the requirement but this is not necessarily true for the loss rate. However, the loss measurement method used for the protocol ensures that the time interval over which the loss rate is computed always includes loss events.

The model is implemented at the receiver. Thus, it is sufficient to report the expected sending rate to the sender. The sender does not need to know model parameters such as the loss rate. These parameters are kept solely at the receiver.

3.3 Estimation of round-trip time and timeout value

The round-trip time t_{RTT} can be measured at the receiver as well as at the sender. The current version of the protocol uses sender based RTT computation. The timeout value t_{RTO} is only used as a parameter for the TCP model and is consequently calculated at the receiver.

3.3.1 Round-trip time measurement

Each data packet carries a timestamp in its header. The receiver sends the timestamp of the last received packet back to the sender in its next report. The sender measures the time when it receives this report and computes the RTT as the difference between the current time t_{now} and the timestamp value t_{ts}.

$$t_{RTTsample} = t_{now} - t_{ts} \tag{3.1}$$

As a receiver report is always generated immediately after a packet arrives, the time between receiving the last data packet with a timestamp and sending the receiver report is negligible compared to the RTT.[1] When a receiver report is lost, the corresponding RTT sample is lost as well, but this has no effect on future RTT samples and does not flaw the RTT estimate.

Single RTT measurements can vary and smoothing of the samples is necessary to prevent unstable protocol behavior. Too much smoothing reduces the responsiveness of the protocol as it takes some time until permanent changes in the RTT are represented adequately in the smoothed RTT estimate. We use an EWMA filter with a low decay factor β as a compromise between stability and responsiveness.

$$t_{RTT} = \beta \cdot t_{RTTsample} + (1 - \beta) \cdot t_{RTT} \tag{3.2}$$

Since the RTT has to be known before the receiver can compute a valid expected sending rate, the sender probes for the RTT by sending packets at a very low rate until it receives the first receiver report.[2]

A different approach is to compute t_{RTT} as the arithmetic average of several RTT samples. However, this does not result in improved protocol behavior. Furthermore, the method requires state information about past RTT values to be kept. For these reasons, the EWMA method is used for the RTT estimate in the TFRC protocol.

3.3.2 Estimation of the timeout value

Instead of computing the timeout value based on the smoothed RTT and the RTT variance as in the TCP protocol, a simple multiple of the smoothed RTT value is used.

$$t_{RTO} = k \cdot t_{RTT} \tag{3.3}$$

[1] If however the sending of the report is delayed for some reason, the delay has to be removed from the RTT computation. This can be done either by directly adding the delay to the timestamp value by decreasing t_{RTT} or by sending the delay back in a report to allow the sender to remove the delay when it determines the RTT.

[2] This state is called *Init RTT* mode. An overview of all protocol modes is given in figure B.1 in the appendix.

This is justifiable since the timeout value is less important to the TFRC protocol than to the TCP protocol. It is not used to schedule the retransmission of packets but merely as a parameter for the TCP model. Its impact on the sending rate is only noticeable under high loss rates.

Due to different clock granularities, the RTT and thus the timeout value can vary considerably from one TCP implementation to another, even under the exact same network conditions. Our estimate of the RTO works well when TFRC is competing against TCP variants with a sufficiently high clock granularity. In environments with a high loss rate, a low RTT, and a coarse RTT estimate of the TCP protocol, the model might overestimate TCP throughput due to its accurate RTT measurement. This unfairness towards TCP is the result of poor TCP behavior and is not a deficiency of the TFRC protocol.

TFRC's aggressiveness can be reduced by introducing a lower bound on the timeout value. The lower bound should be equal to the smallest possible timeout value of competing TCP variants. For coarse granularity TCP clocks, the value is in the range of 500 ms to one second.

3.4 Estimation of the loss event rate

A loss event occurs when there is not enough bandwidth available along the path from sender to receiver and packets have to be discarded. The ratio of losses compared to successfully transmitted packets reflects the level of congestion. Using loss as an indication of congestion implies that the number of packets lost due to network errors must not exceed a small fraction of packets dropped by congested routers.

As required by the TCP model, TFRC does not measure the packet drop rate[3] but determines the loss event rate. A loss event is composed of one or more lost packets during one RTT. The definition of a loss event corresponds to the maximum of one window reduction event per RTT by many TCP variants. It also ensures that the sender has time to respond to a loss indication by the receiver before further loss events are indicated. The packet drop rate serves as an upper bound on the loss event rate. In order to accurately render the current network conditions, it is important that the receiver uses the current RTT t_{RTTcur} and not the smoothed RTT to determine which lost packets form a loss event.

Packets are marked with consecutive sequence numbers. The receiver notices lost packets by detecting flaws in the order of sequence numbers. To account for packet reordering which also manifests itself as a flaw in the order of sequence numbers, a method similar to TCP's triple duplicate ACK mechanism is used to distinguish lost packets from reordered packets. While it is impossible to ascertain that a packet is lost and not just delayed, reordering of a large number of packets is improbable in most networks. It is important to detect loss events as early as possible to guarantee sufficient responsiveness to congestion. The protocol waits for three more packets after receiving an out of order sequence number. If all three sequence numbers are higher than the missing sequence number, the packet with the missing sequence number is assumed lost and the estimate of the loss event rate is adjusted accordingly. Out of order reception of up to three data packets is handled correctly.[4] Duplicate packets are ignored by the loss measurement algorithm.

The loss event rate always has to be measured over a certain time interval, called the *loss history*. How to

[3] The packet drop rate is the number of lost packet over the total number of transmitted packets.

[4] When missing packets arrive after more than three intermediate packets, the loss event is not revised. Major packet reordering is uncommon for most computer networks and it is safe to be conservative and reduce the sending rate under these circumstances.

determine the loss history is the main focus of the remainder of section 3.4. There is a clear tradeoff between responsiveness by using a short measurement period and stability by using long measurement period. The loss measurement method is crucial for good protocol behavior since the fair share of bandwidth for a flow is mostly determined by the loss event rate. This module of the protocol has undergone a very thorough evaluation and its design has been improved several times during the development process. Two important characteristics for the loss estimation method can be identified. The loss estimate should adjust smoothly to a steady packet drop rate with approximately evenly spaced loss events. A ramp up in the loss rate each time a loss event occurs in steady state should be prevented. Furthermore, the loss estimate should respond strongly to several consecutive losses since this indicates severe congestion.

3.4.1 Loss measurement methods

The investigated loss measurement methods either directly compute the estimated loss event rate l_{est} or determine the average number of packets between loss events ppl, called loss interval. These two are equivalent, since $l_{est} = 1/ppl$. The size of the loss history is measured either in packets or in number of loss intervals depending on the measurement method.

Dynamic loss window

An obvious method to determine the loss event rate is to measure the number of loss events over a fixed number of packets using a sliding window technique. This method requires little effort to implement but it has several drawbacks. As stated in the introduction, the TFRC protocol is designed to work under a wide range of network conditions and to be able to cope well with heterogeneity. Using a fixed number of packets is not flexible enough for this purpose. A large value for the window size causes the protocol to adapt only very slowly to changes in the loss event rate at low sending rates while a small value causes the protocol to react even to small amounts of noise at high packet rates. Thus, the size of the loss window has to be dynamically adjusted to the current conditions.

In the first approach, the window size was inversely proportional to the loss event rate and thus proportional to the sending rate. The receiver uses the inverse of the TCP throughput equation 2.7 to determine the loss rate l_{exp} that is expected, given the current sending rate and RTT. l_{exp} is then used to determine the size of the loss window h.

$$h = \frac{K}{l_{exp}} \approx K \cdot ppl \tag{3.4}$$

K is the number of loss events needed for a meaningful sample of the loss event rate. The loss estimate is independent of the sending rate.

This method lacks several important characteristics:

- Due to changes in the loss rate, the loss window may not include any loss events at all and $l_{est} = 0$. A loss window without any loss events cannot be used to estimate the loss event rate. Furthermore, applying the TCP equation on a loss event rate of zero results in an infinite expected bitrate.[5]

[5] One could argue that a loss estimate of zero indicates that there is no congestion and thus the sending rate should be increased with the maximum possible increase factor until a loss event occurs. However, this approach causes instabilities in the sending rate and is very susceptible to a noisy packet drop rate.

- In some cases, most of the loss events will be at the beginning of the loss window (i.e. the part of the window where old packets are excluded when the window moves on). When the sending rate decreases, the loss window size will decrease as well, possibly excluding these loss events. Without the loss events, the loss event estimate decreases which in turn increases the sending rate and loss window size. The loss events may be included again, then excluded due to the subsequently increased loss estimate, etc. This leads to oscillations of the sending rate and very unstable protocol behavior.

To overcome the first problem, a lower bound on the number of loss events in the loss window is introduced. If the loss window contains less than E loss events, the window size is increased until the number of loss events is equal to E. The second problem is avoided by introducing a bound on the start of the loss window. By setting this bound to the most recently excluded packet, the loss window size can only increase when new packets arrive, not by including previously excluded packets. In particular, discarded loss events cannot be included in the loss window again.

While these additions are necessary for the deployment of this method, they are not sufficient for stable protocol behavior.

- Consider again a number of loss events at the beginning of the loss window. When a new loss event occurs, the loss estimate increases and the loss window size is reduced. This can cause the loss events to be excluded. Thus, a loss event can actually *decrease* the loss estimate.
- The number of loss events in the loss window has to be relatively small to ensure acceptable protocol responsiveness at common loss rates of one loss event per hundreds or thousands of packets. Losses entering or leaving the loss window change the loss estimate considerably. When K is set to 4, a loss event entering the loss window changes the loss rate by 20%.
- The exclusion of old packets or loss events at the start of the loss window should have less influence the loss estimate than the inclusion of new loss events. Weighting these recent events more cannot easily be accomplished with this method.
- The feedback loop inherent in this method is problematic. The loss estimate l_{act} is used to determine the expected sending rate. This will be the rate that the sender uses after one RTT. Thus, the inverse equation applied to the sending rate will lead to a value of l_{exp} that equals the previous value of l_{act}. This dampens the window size adjustments in the form of a time delay of one RTT.[6] While damping as such is necessary, better forms of damping can be used in conjunction with other loss estimation methods.

The many necessary additions and unresolved problems with window-based loss estimation lead us to believe that this method is not very well suited for the task.

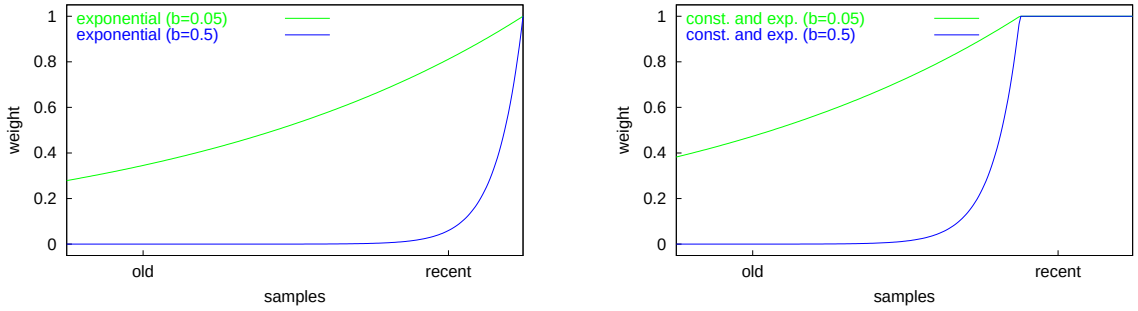
Exponential moving average

A simple and straightforward loss estimation method is to use the EWMA of the number of packets between loss events. When loss event i occurs, the corresponding loss interval ppl_i is added to the exponential moving average.

$$ppl_i^{EWMA} = \gamma \cdot ppl_i + (1 - \gamma) \cdot ppl_i^{EWMA} \tag{3.5}$$

[6] Directly using the previous l_{act} to determine the window size is not advisable since it lacks any form of damping.

However, this method also has several drawbacks. Setting γ to a value that works well for a wide range of packet loss probabilities proves to be very difficult. Figure 3.2a) shows loss interval weights for two different values of γ. A large γ value of for example 0.5 leads to rapidly decaying weights for previous loss intervals. The loss interval estimate ppl_i^{EWMA} consists almost solely of the last loss interval. While this results in fast adaption to changed network conditions, it is also very susceptible to noise. A small γ on the other hand puts too much weight on old loss intervals and adapts only slowly to changes in the level of congestion.



a) Exponential weights

b) Combined constant and exponential weights

Figure 3.2: Exponential weight distribution

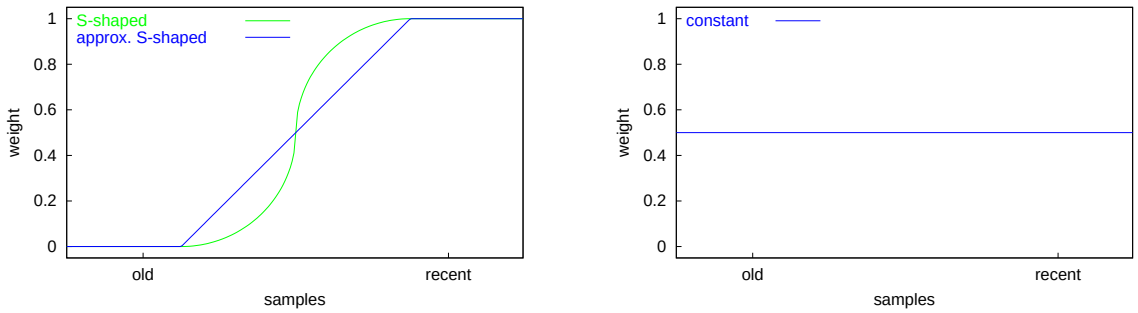
The first problem can be alleviated by using a combination of constant and exponential weights. The most recent loss intervals receive equal weights while the weights for older loss intervals decay exponentially as depicted in figure 3.2b). It is easy to ensure that the transition from constant to exponential weights is continuous, however it cannot be continuously differentiable. When a new loss event occurs and the weight for an interval switches from constant to exponential, the change between the old and the new weight for the loss interval is substantial. That leads to unneccessary discontinuities that can cause unstable protocol behavior.

Weighted loss interval average

An alternative to using an EWMA is to use the arithmetic average of n loss intervals. Since the number of loss intervals is fixed, the loss rate is determined by the number of packets in these intervals instead of the number of loss events. This allows for results in fine grained changes to the loss estimate.

$$ppl_{avg,i} = \frac{\sum_{j=0}^{n-1} w_j \cdot ppl_{i-j}}{\sum_{j=0}^{n-1} w_j} \tag{3.6}$$

As long as the total number of loss intervals is less than n, the total number of loss intervals is used to compute the loss estimate. The simple approach of using equal weights for all of the intervals (figure 3.3b)) is not useful. Loss events leaving the loss history would affect the loss estimate to the same extent as loss events entering it. An *s-shaped* weight distribution is better suited for the task. Recent loss intervals receive similar weights since they represent the current state of congestion in the network. There is a smooth decline in weighting after a certain number of intervals to ensure that older loss events, and especially loss events that leave the loss history, do not affect the loss estimate. In the current version of the protocol, the s-shape



a) S-shaped weights

b) Constant weights

Figure 3.3: Weighted loss interval average

of the weights is linearly approximated as depicted in figure 3.3a). The weights for the $n/2$ most recent loss intervals are set to one, while the remaining $n/2$ weights decrease towards zero. All older loss intervals receive a weight of zero.

Because of its advantages, the Average Loss Interval Method is currently used for loss rate estimation in the TFRC protocol. The loss history size is set at eight loss intervals, which is a reasonable compromise between stability and responsiveness. Simulations that corroborate these findings are presented in chapter 4.8.1 and 4.10. Table 3.1 shows distribution of the weights.

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
1.0	1.0	1.0	1.0	0.8	0.6	0.4	0.2

Table 3.1: Weight distribution for the loss intervals

3.4.2 Steady state

Assume a steady state with a constant RTT and a steady packet drop rate of one packet every m data packets. The loss estimate of TFRC remains constant leading to a constant sending rate. In contrast, TCP halves its sending rate due to a loss event and subsequently ramps up again until it experiences the next loss event, as depicted in figure 3.4. Equidistant periodic packet loss usually never occurs in “real” networks due to a certain amount of noise present in the network. Nevertheless, this analysis emphasizes the different stability characteristics of TFRC and TCP in a rather steady environment.

3.4.3 Accounting for the current loss interval

All loss interval-based methods have to deal with the problem that the most current interval usually does not end with a loss event. If this interval is included, it may not represent the underlying packet drop rate. When a loss event occurred recently, the number of packets since the last loss event is much smaller than the average loss interval size. Including this interval leads to an overestimation of the loss event rate. However, not including the current interval until it ends with a loss event is not a viable solution either.

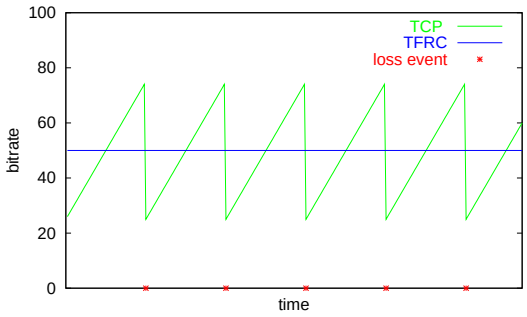


Figure 3.4: Steady state sending rate

When the packet drop rate decreases, it takes a long time until new loss events occur and the loss estimate is updated. Responsiveness would decrease considerably with this method. In particular, problems arise in an environment with low statistical multiplexing (i.e. with few competing flows). When the competing flows stop sending, the bandwidth utilization falls below the bottleneck bandwidth and no more loss events occur at all. Thus, the loss estimate would *never* be updated and the sender would continue to send at a constant rate.

An elegant way to avoid these problems is to include the current loss interval only when this causes the loss estimate to decrease. Figure 3.5 shows the conventional loss estimates with and without the current interval in comparison to the improved loss measurement method, assuming a steady state with alternating large and small loss intervals. As can be seen from figure 3.5b), including the current loss interval only when its size exceeds the average loss interval size results in a loss estimate which is the minimum of the other two estimates.[7]

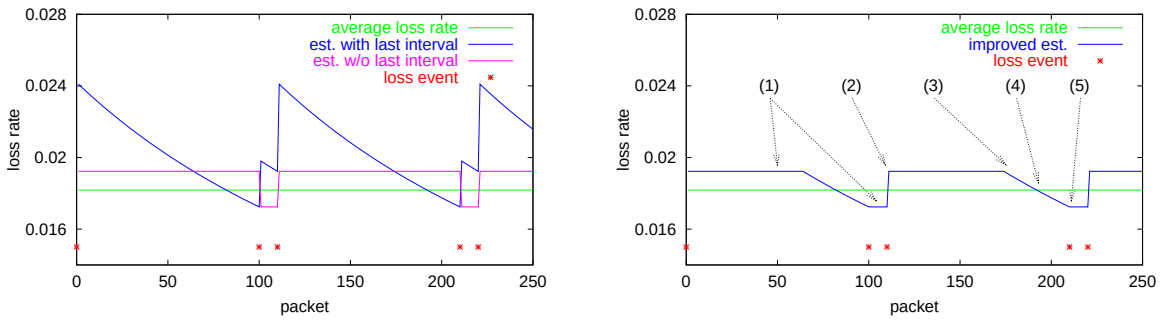
The improved method has two states:

***INC*:** As long as including the most recent interval (not ending in a loss event) would reduce the loss estimate, the n most recent loss intervals *without* the current interval are used and the loss estimate does not change (figure 3.5b), (1)). When a loss event occurs in this mode, thus ending the current loss interval, the interval has to be included in the loss estimate and causes the estimate to *increase* (figure 3.5b), (2)).

***DEC*:** When no loss event occurs during *INC* mode, the current loss interval will eventually exceed the average loss interval size. When this happens, the current interval is included in the loss estimate, providing a smooth transition between the two modes (figure 3.5b), (3)). Now, each new packet arrival increases the size of the current interval and the loss estimate has to be updated (figure 3.5b), (4)). Eventually, a loss event will occur in this mode. Since it is already anticipated by the inclusion of the current interval, the loss estimate does not change (figure 3.5b), (5)). The mode is changed back to *INC*.

To prevent discontinuities in the transition from *DEC* to *INC* mode, it is important to "preshift" weights at this point. Although the current loss interval does not yet end with loss event $i+1$, the weight w_j previously used for loss interval ppl_{i-j} is now used for interval ppl_{i-j+1}.

[7] Note that both figures use packets as the unit for the x-axis instead of time. Thereby, the loss estimate becomes independent of the sending rate (i.e. the packet arrival rate).



a) Loss estimate with/without last interval

b) Improved loss measurement method

Figure 3.5: Loss estimate

3.5 Changing the sending rate

When the RTT and/or the loss estimate change, the receiver has to update the expected bitrate and report it back to the sender. Consequently, the sender will adjust the sending rate. To directly use the new rate is not advisable since the altered bitrate might cause changes in other network parameters. For instance, an increased sending rate can cause buffers to fill up which leads to an increased round-trip time. This in turn results in a decrease in the expected bitrate. To prevent such oscillations, the bitrate has to be adjusted gradually so that the receiver has time to report the new network conditions back to the sender.

3.5.1 Increase policy

When the expected bitrate b_{exp} exceeds the current sending rate b_{act}, the current sending rate is increased. The increase value Δb is set to one packet per RTT (i.e. an increase of at most one packet per RTT over a time interval of one RTT). This bounds TFRC's aggressiveness by the TCP bitrate increase in the absence of packet loss.

$$\Delta b = \frac{s}{t_{RTT}}, \text{ where } s \text{ is the packet size} \tag{3.7}$$

$$b_{act} = \min(b_{act} + \Delta b, b_{exp}) \tag{3.8}$$

Whenever the sending rate is higher than one packet per RTT, the rate is adjusted smoothly by performing a partial increase each time a packet is sent. This results in a gradual decrease in the time interval between two consecutive data packets. Data packets are never sent back-to-back (i.e. there is always a small gap before the next packet is sent). The partial amount of increase Δb_P is computed as the total increase Δb over the number of packets n_{RTT} sent during the next RTT. n_{RTT} and Δb_P are recalculated every time a report is received.

$$n_{RTT} = \frac{b_{act} + \Delta b}{2} \cdot \frac{t_{RTT}}{s} = \frac{b_{act} \cdot t_{RTT}}{2\,s} + \frac{1}{2} \tag{3.9}$$

$$\Delta b_P = \frac{\Delta b}{\max(n_{RTT}, 1)} \tag{3.10}$$

When the sender does not receive reports on time, the network conditions might have changed in the meantime and the expected bitrate is no longer valid. While one missing receiver report or a slight delay are

common during normal operation, more missing reports or a longer delay indicate an increased congestion level. To prevent a further rate increase under these circumstances, b_{act} is bounded by b_{max}.

$$b_{max} = b_{act} + 2\,\Delta b \tag{3.11}$$

When a receiver report arrives, the sender updates b_{max}. Thus, the bound of b_{max} has no effect when a report is delayed by up to one RTT or if a report is lost. A larger delay or several lost reports prevent the sending rate from further increasing after one RTT. When this condition persists, the sending rate eventually has decrease.

3.5.2 Decrease policy

Several alternatives exist on how to decrease the sending rate, when the current sending rate exceeds the expected bitrate.

- decrease to b_{exp}
- decrease towards b_{exp} (without fully decreasing to b_{exp})
- decrease to $1/2\,b_{act}$ (exponential decrease)

Decreasing the bitrate by half even when it only slightly exceeds the expected bitrate impairs the protocol stability and makes the protocol very susceptible to noise.[8] Undershooting the expected bitrate by a large amount will subsequently cause the sending rate to increase until the expected bitrate is reached again. This form of rate control is too oscillatory for most applications.

Sufficient damping is offered by using an EWMA for the RTT and a weighted average for the loss estimate. Thus, additional damping by reducing the bitrate only by a fraction of the expected decrease is not necessary. Too much damping prevents a timely response to congestion. The policy chosen for the TFRC protocol is to not decrease towards b_{exp} but to b_{exp} all at once.

Timeout of receiver report

The TCP congestion window limits the number of outstanding packets in the network, called the "conservation of packets" principle [Jac88]. This results in a closed-loop control system. When no more acknowledgements are received by the sender, it has to stop sending packets after the congestion window is used up. In contrast to TCP, rate-based congestion control is an inherently open-loop control system. The sender just continues to send at a given rate in the absence of receiver reports.

Missing or delayed reports can be caused by a high level of congestion or when the receiver crashes or becomes unreachable. In all of these cases, it is important that the sender reduces its rate and eventually ceases to send when the condition persists. A timer with a timeout value t_{dec} set to a multiple of the RTT is used to detect missing reports.[9]

$$t_{dec} = d \cdot \max\left(t_{RTT}, \frac{s}{b_{act}}\right) \tag{3.12}$$

[8] This form of rate halving is not equivalent to TCP's exponential decrease. Here, the rate is halved whenever the expected bitrate falls short of the actual sending rate, for example because of a slight increase in RTT. TCP halves the rate only in response to a loss event.

[9] When the sending rate is less than one packet per RTT, the time between two consecutive packets is used instead of the RTT.

When no report is received during this interval, the protocol assumes severe congestion. It reduces its sending rate by half and resets the timer. When no reports are received in the next time interval, the rate is halved again, resulting in an exponential rate decrease. As soon as receiver reports arrive at the sender again, the sending rate can be readjusted to the rate expected by the receiver. It is not necessary to invalidate the history at the receiver after such an event. The receiver can still keep track of arriving and missing packets and compute a valid loss estimate.

In practice, values for d in the range from five to ten provide good results. The multiplier d must not be much lower. The protocol shouldn not respond to one or two dropped receiver reports by halving the sending rate.

3.6 Other protocol features

3.6.1 Slowstart algorithm

Like TCP, we employ a *slowstart* algorithm to determine the available bandwidth when the sender to starts transmit packets. When the sender receives the first receiver report and thus has a valid sample of the RTT, it switches to slowstart mode. Starting with a sending rate of one packet per RTT, the rate is doubled every RTT to quickly reach the fair share of bandwidth.

$$b_{exp} = 2\, b_{act} \tag{3.13}$$

The rate increase is not limited by a maximum of one packet per RTT during slowstart mode. Like normal rate increase, slowstart increase is done smoothly by adjusting the inter-packet spacing over the number of packets sent during the next RTT.

As soon as the receiver encounters the *second* lost packet, a report is immediately sent back to the sender which then quits slowstart mode. The first lost packet[10] is not taken into account to be less susceptible to premature termination of slowstart mode due to random loss. When the link bandwidth is reached, several packets will be lost and slowstart mode terminates correctly. The sending rate of the protocol when slowstart is denoted with b_{sst}.

TCP's slowstart mechanism prevents reaching a sending rate of more than twice the bottleneck bandwidth. Under normal conditions, the TFRC slowstart mechanism will behave similarly. A loss will occur when the bottleneck bandwidth is reached. It then takes one RTT until the sender receives a report reflecting that loss event. In the meantime, the rate will be doubled once, resulting in $b_{sst} = 2\, b_{act}$.

However, this is not the case when the connection from sender to receiver is overbuffered. Consider an example where buffering of a link is a multiple of the bandwidth delay product[11] and no competing traffic is present. When the bottleneck bandwidth is reached, the buffer starts filling up. After one RTT, the buffer comprises one bandwidth delay product of data and the sending rate is $2\, b_{lnk}$. After two RTTs, the buffer comprises four bandwidth delay products and the sending rate is $4\, b_{lnk}$, etc. This continues until the buffer is full and the first packets are dropped. To prevent a rate overshoot to more than twice the link bandwidth, the sending rate has to be limited explicitly. The receiver gives feed back about the rate b_{rep} with which it is

[10] not *loss event*, where potentially several lost packets during one RTT are aggregated to form one loss event

[11] The bandwidth delay product is defined in chapter 2.1.2.

actually receiving the data. Since this rate cannot exceed the bottleneck bandwidth, it is used to bound the rate increase.[12]

$$\Delta b = \min(b_{act}, b_{rep}) \tag{3.14}$$

$$b_{exp} = b_{act} + \Delta b \tag{3.15}$$

Again, we have to account for missing receiver feedback. Because of the fast increment of the sending rate during slowstart, the timeout interval for missing reports has to be lower than the timeout used during normal protocol operation. Setting the slowstart timeout t_{dec}^{ss} to half the normal timeout interval by using a reduced multiplier d' provides good results.

$$t_{dec}^{ss} = d' \cdot t_{RTTsample} \text{ , where } d' = \frac{1}{2}d \tag{3.16}$$

A second modification is the use of the current RTT sample $t_{RTTsample}$ instead of the smoothed average t_{RTT}. The RTT EWMA always lags behind when the RTT increases monotonically. This is often the case during slowstart due to an increased buffer utilization. The current RTT samples provide a more accurate value. When the timer triggers, the TFRC protocol terminates slowstart and switches to normal congestion control mode. The timer is not reset. As a result, the rate halving timer for the congestion control mode triggers after the time interval t_{dec} when no reports arrive in the meantime. When a receiver report eventually arrives and does not indicate any loss events, the sender reenters slowstart mode. If the report indicates a loss event, slowstart is terminated for good and the sender continues to operate in congestion control mode.

3.6.2 Loss history initialization

When slowstart mode is terminated, the loss event rate has to be initialized. The number of packets transmitted during slowstart does not reflect the underlying packet drop rate of the connection and cannot be used to estimate the loss event rate. Because of the rapid rate increase, the number of transmitted packets is lower than what would have been sent if the sender had constantly transmitted at rate b_{sst}. At the same time, it is likely that a loss event would have happened sooner at such a high sending rate.

The slowstart rate is limited to twice the bottleneck bandwidth. Thus, we can assume that the bottleneck bandwidth is $1/2\, b_{sst}$ and the receiver should report this rate as the first expected bitrate. To seed the loss history accordingly, the receiver computes a loss rate l_{exp} corresponding to $1/2\, b_{sst}$ by means of the inverse equation of the TCP equation 2.7. The first loss interval ppl_0 is then initialized to $1/l_{exp}$. By only initializing the first loss interval instead of all n intervals, the protocol makes sure that the loss estimate adapts quickly when "real" loss intervals become available. The synthetic loss interval is aged like normal loss intervals and will eventually be discarded from the loss history after n further loss events.

3.6.3 Treating send errors

Send errors occur, when the sender cannot send a packet because all buffer space of the operating system is used up. These errors occur frequently instead of packet losses when sending at a high speed to a receiver

[12] Usually, it is sufficient to just use b_{rep} instead of $\min(b_{act}, b_{rep})$, because the receiving rate is always less than or equal to the sending rate. However, it is difficult to measure b_{rep} correctly during slowstart because of the fast rate changes. Using the minimum of the two rates protects against errors in the receiving rate measurement.

on the same Local Area Network (LAN). Especially during slowstart, the sender can cause the send buffer of the operating system to overflow. In that case, the sender does not try to retransmit the packet. It is thus treated as a normal packet loss by the receiver. The fact that the sender knows beforehand that the receiver will experience a loss is useful in slowstart mode. The sender can terminate slowstart immediately without waiting for the receiver to report the loss and prevent congestion in the network. This is done by switching to *force congestion control* mode[13]. In this mode, the sender cannot switch back to slowstart when newly arrived reports indicate no packet loss. Receiver reports are discarded until they reflect a positive loss estimate caused by the packet that was not sent because of the send error.

3.6.4 Inter-packet space modulation

The delay in the feedback loop can cause sending rate oscillations when very few connections share the same link. The network conditions do not comply with the model assumptions of an independent RTT. Consider an example where a TFRC flow is the only flow on an otherwise empty link with sufficient buffering. In the absence of loss events, the TFRC flow increases its rate. When the rate reaches the link bandwidth, the buffer starts to fill up and the RTT increases by the buffer delay. Eventually, the buffer will flow over and packets have to be dropped. The loss rate increases and the sender reduces its sending rate. This causes the buffer to empty and the buffer delay to decrease, which in turn leads to a rate increase. The result is a cycle of increase and decrease instead of a steady sending rate in such an environment.

The increase in delay should cause the sender to dampen the increase when the buffer starts to fill up. Equally, when the sending rate is decreased due to a loss event and the buffer empties, the reduced buffer delay should dampen the decrease. However, the RTT variations do not reduce rate changes but on the contrary introduce further oscillations because of the delay in the RTT computation. It takes one RTT until the sender gets a new RTT sample and another half RTT until the receiver is notified of the change. Furthermore, the RTT is smoothed using an EWMA filter. The resulting overall delay prevents damping of the oscillations. Reducing the smoothing of the RTT is not an option since that increases the protocols susceptibility to noise. Also, as mentioned in chapter 2, the TCP model uses a long-term notion of RTT and an RTT that reacts to changes in the buffer level does not comply with the model. A solution to this problem is to modulate the inter-packet spacing according to the ratio of current RTT and smoothed RTT, while retaining the long-term notion of the RTT as a model parameter. The time interval Δ_p between two consecutive data packets at the current sending rate b_{act} is given as

$$\Delta_p = \frac{s}{b_{act}} \tag{3.17}$$

An RTT sample that is higher than the average RTT indicates that a buffer is filling up. In that case, the time interval between the packets should be increased. Likewise, this interval should be decreased when the RTT samples are less than the average RTT. We do not use the ratio of the values directly to modify the inter-packet spacing, but the ratio of the square roots. Using the ratio directly modifies the inter-packet spacing and thus the sending rate to such an extent, that this in itself can lead to oscillations instead of removing them. The time interval between packets Δ_p^{ISM} using Inter-packet Space Modulation (ISM) is calculated as folows:

$$t_{RTT}^{sqrt} = \beta \cdot \sqrt{t_{RTTsample}} + (1-\beta) \cdot t_{RTT}^{sqrt} \tag{3.18}$$

[13]For a complete overview of the protocol states see appendix B

$$f_{ISM} = \frac{\sqrt{t_{RTT\,sample}}}{t_{RTT}^{sqrt}} \tag{3.19}$$

$$\Delta_p^{ISM} = \frac{s}{b_{act}} \cdot f_{ISM} \tag{3.20}$$

In addition to the smoothed RTT, the smoothed average of the square root of the RTT samples t_{RTT}^{sqrt} is computed from each RTT sample, using the same decay factor as the factor used for the smoothed RTT estimate. The ISM factor f_{ISM} is the square root of the current sample over the average square root RTT. Δ_p^{ISM} is then used instead of Δ_p for the timing of the data packets.

Modifications in the sending rate by ISM will cancel itself out over time, since the RTT samples form the smoothed RTT estimate. In a simplified way, for each RTT sample that is smaller than the average RTT, there has to be an RTT sample that is higher than the average RTT. The long-term sending rate is unaffected by ISM. Experiments with and without ISM are presented in chapter 4.3.

When the number of flows that share the same link is high, the behavior of a single flow has only a very small impact on the RTT. The RTT samples remain relatively constant and the factor for the inter-packet spacing has little effect.

3.7 Additional deweighting of unrepresentative loss intervals

In most cases, linearly decreasing weights for older loss intervals put sufficient weight on the more recent intervals. Yet, it might not be sufficient when no loss events occur for a long time interval. In this case, the duration or size of the current loss interval can be disproportionately higher than the average of the other intervals. It is questionable that the older loss intervals represent the current loss event rate under these circumstances. While the loss estimate decreases with the arrival of new packets, it does so very slowly since too much weight lies on unrepresentative intervals.

Figure 3.6 shows the weight distribution of a typical TFRC session. The width of the loss intervals corresponds to their size and the height corresponds to their weight in the average loss interval computation.

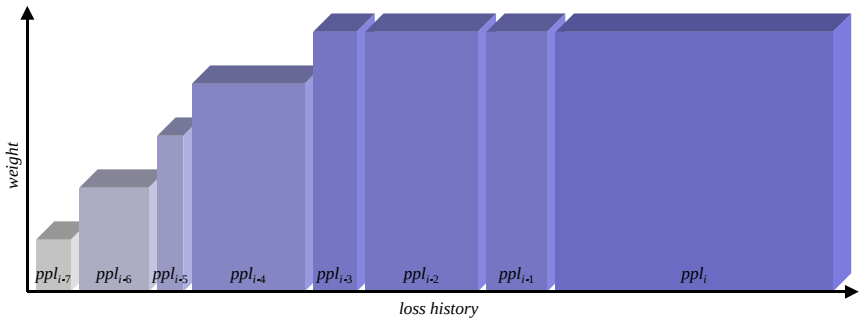


Figure 3.6: No deweighting

Loss intervals are considered unrepresentative of the loss event rate, when the size of the current interval ppl_i exceeds a multiple m of the average size ppl_{i-1}^{avg}. Three methods for additional deweighting are investigated.

3.7.1 Discarding of old intervals

The easiest way to deweight old loss intervals is to discard them completely. A possible approach is to reduce the number of loss intervals in the loss history according to the ratio of the current loss interval size to the average loss interval size. Let n be the number of loss intervals in the loss history during normal operation and n' be the number of loss intervals for deweighting. n' can be computed as follows:

$$n' = \max\left(n - \left\lceil \frac{ppl_i}{m \cdot ppl_{i-1}^{avg}} \right\rceil, 1\right) \tag{3.21}$$

In this context, m determines when and how fast old intervals are deweighted. The more loss intervals are discarded, more the loss estimate changes when the next interval is removed from the loss history. These sudden changes impair protocol stability. Since stability is one of the main concerns of the protocol, this alternative was not pursued any further.

3.7.2 Proportional Deweighting

For *Proportional Deweighting*, a deweighting factor d_i is determined for the unrepresentative intervals to reduce their impact on the loss estimate.

$$d_i = \begin{cases} 1 & \text{for } ppl_i \leq m \cdot ppl_{i-1}^{avg} \\ \frac{m \cdot ppl_{i-1}^{avg}}{ppl_i} & \text{for } ppl_i > m \cdot ppl_{i-1}^{avg} \end{cases} \tag{3.22}$$

The greater the discrepancy between the average loss interval size and the current loss interval size, the smaller the deweighting factor and thus the weight for older loss intervals.

$$ppl_{i-1}^{avg} = \frac{w_0 \cdot ppl_i + d_i \sum_{j=1}^{n-1} w_j \cdot ppl_{i-j}}{w_0 + d_i \sum_{j=1}^{n-1} w_j} \tag{3.23}$$

The modified weights after proportional deweighting are depicted in figure 3.7.

Figure 3.7: Proportional deweighting

3.7.3 Loss Interval Duplication

A different method to provide additional deweighting during long periods without losses is to duplicate the last interval once its size exceeds a threshold. For duplication, the current interval ppl_i is inserted as a

new loss interval, as shown in figure 3.8. Thus, the two most recent intervals have the same size and the oldest interval is discarded from the loss history. A new average $ppl_{avg,i+1}$ is computed using the duplicated interval. Packets arriving after the duplication increase the size of the new current interval ppl_{i+1}.

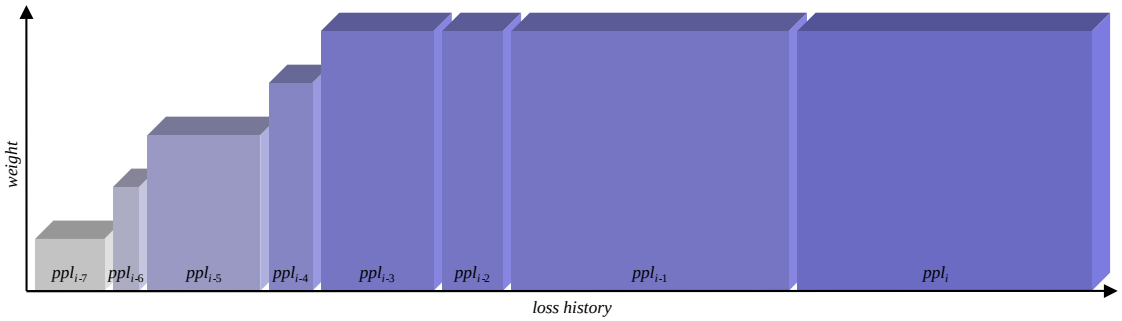


Figure 3.8: Deweighting by loss interval duplication

3.7.4 Comparison

The deweighting is done more smoothly by the *Proportional Deweighting* method. Weights can be modified by very small amounts, which results in small changes to the loss estimate. *Loss Interval Duplication* can modify the loss estimate considerably, when the duplicated interval is much larger than the old interval that is discarded. Furthermore, the number of packets in the loss history does no longer correspond to the actual number of received packets.

However, when deweighting is suspended because the frequency of losses increases again, the *Loss Interval Duplication* method shows better stability characteristics. The loss history is composed of loss intervals with gradually increasing sizes because of the duplication process. New loss intervals change the loss estimate only by a relatively small amount. Using the *Proportinal Deweighting* method results in a loss history where most of the weight lies on the current interval. A following small interval causes an large increase in the loss rate as the rate is basically computed over two intervals instead of n.

It is desirable to combine the good features of both deweighting methods. There are two ways to modify proportional deweighting to improve its stability in case of an increase in the loss rate after deweighting.

- Imposing a lower bound d_{lim} on the deweighting factor prevents that all the weight is shifted to the current loss interval.
- When deweighting is suspended because a new loss event occurs, the loss estimate can be used to *remodel* the loss history. The average loss interval size is given by the inverse of the current loss estimate. By setting the size of each loss interval in the loss history to the average size, the loss estimate remains the same, while the deweighting factor is no longer necessary. The original weights for the loss intervals can be used and the impact of new loss intervals on the loss estimate is proportional to the size of the loss history.[14]

The former alternative limits the amount of deweighting which impairs responsiveness when the loss rate

[14]While a lower bound for the deweighting factor is no longer necessary, it may be useful to keep at least a small amount of "old" history in the loss estimate. This results in slightly less aggressive protocol bevahior.

changes from a very high to a very low value. This is not true for the latter approach. In the extreme case, all the weight can be shifted to the last interval while the method still provides a stable loss rate increase after deweighting is suspended. Simulations as well as experiments that corroborate these findings are presented in section 4.8.1. The TFRC protocol uses proportional deweighting with loss history remodeling for the loss estimate.

Chapter 4

Network Experiments and Simulations

To assess how the TFRC protocol performs in todays networks (LANs, WANs, Internet), protocol simulations *and* experiments are crucial [PF97]. In this chapter, we discuss the results of a number of network experiments conducted with the TFRC protocol. Some simulation results are presented in order to point out certain aspects of protocol behavior that are difficult to study in uncontrolled environments as the Internet. In-depth simulation studies are presented in [Pad00].

4.1 Performance metrics

First, we introduce a notion of throughput. Based on this definition, we define inter- and intra-fairness metrics as well as metrics to analyze protocol dynamics over time.

4.1.1 Throughput

With $n_{f,\delta}(t)$, we denote the number of packets of a flow f during a certain time interval of duration δ and ending at time t. If the amount of time since the flow started is less than δ then $n_{f,\delta}(t)$ is defined as the total number of packets of the flow f since the start time t_0.

$$n_{f,\delta}(t) = \begin{cases} \text{\# packets sent in } (t-\delta, t] & \text{for } t-\delta \geq t_0 \\ n_{f,\delta}(t) = n_{f,t-t_0}(t) & \text{for } t-\delta < t_0 \end{cases} \tag{4.1}$$

The throughput $B_{f,\delta}(t)$ of a flow at time t with granularity δ and with a packet size of s can be obtained by measuring the amount of transmitted data over a time interval of size δ.

$$B_{f,\delta}(t) = \frac{n_{f,\delta}(t) \cdot s}{\delta} \tag{4.2}$$

Because the throughput depends on both the frequency of the packets and the packet size, the same throughput can be achieved by different combinations of the two parameters (i.e. by a small number of large-sized packets per time interval or a great number of small-sized packets). The granularity of the throughput metric can be set by modifying the parameter δ. The larger the value of δ is, the lower is the granularity. It is

important not to choose the value for δ too small (i.e. a high granularity) when the packet frequency is low. This ensures that the number of packets per time interval adequately represents the throughput.

The average throughput B_f of a flow is defined as the total amount of data being transmitted during the whole measurement interval $[t_0, T]$ over time.[1]

$$B_f = \sum_{i=1}^{(T-t_0)/\delta} \frac{n_{f,\delta}(t_0 + \delta i) \cdot s}{T} = B_{f,T-t_0}(T) \tag{4.3}$$

To be able to compare the throughput of different types of protocols, we introduce protocol throughput B_P as the average throughput of the set of flows P that use this protocol. $|P|$ is the number of flows that utilize the protocol P.

$$B_P = \frac{\sum_{f \in P} B_f}{|P|} \tag{4.4}$$

We introduce normalized throughput in order to analyze the throughput distribution of different flows. Normalized throughput is defined as

$$B_f^n = \frac{B_f}{B_P} \tag{4.5}$$

The unit of throughput used in this chapter is KByte/s unless noted otherwise. The normalized throughput is dimensionless.

4.1.2 Inter-protocol fairness

Let $\mathcal{P}$ be the set of all the flows of an experiment. To measure how fair flows of a certain protocol $P \subseteq \mathcal{P}$ behave against competing flows, the *inter-protocol fairness* measure is introduced. The inter-protocol fairness of a protocol is defined as the ratio of the average throughput of all competing flows and the sum of the average throughput of protocol flows and competing flows.

$$F_P^{\text{inter}} = \frac{B_{\mathcal{P}\setminus P}}{B_P + B_{\mathcal{P}\setminus P}} = 1 - \frac{B_P}{B_P + B_{\mathcal{P}\setminus P}} \tag{4.6}$$

A protocol is fair, when $F_P^{\text{inter}} \geq 0.5$. A fairness value of 0.5 implies that flows have the same average throughput no matter if they operate according to protocol P or a different protocol. An inter-protocol fairness of 0.5 is called *ideal fairness*. Fairness values greater than 0.5 indicate too conservative protocol behavior. Flows using that protocol claim less than their fair share of bandwidth. In the extreme case, these flows use no bandwidth at all, which results in a fairness value of one. A protocol operates too aggressive, when its fairness value falls short of 0.5. Here, the extreme case is a fairness value of zero, where no bandwidth is left for flows not using the investigated protocol. The inter-protocol fairness measure is independent of the number of flows in P and $\mathcal{P}$.

4.1.3 Intra-protocol fairness

We define two intra-protocol fairness metrics to capture different aspects of fairness. The common concept of *max-min fairness* [BG87] is used to estimate the throughput range of the flows of a protocol. It is defined

[1] B_f is independent of δ.

as the ratio of the minimum average flow throughput and the maximum average flow throughput.

$$F_P^{\text{max-min}} = \frac{\min_{f \in P} B_f}{\max_{f \in P} B_f} \tag{4.7}$$

The max-min fairness measure lies in the interval $[0, 1]$. A fairness value of zero indicates, that *at least* one flow received no bandwidth at all. A value of one is achieved in case of an equal distribution of bandwidth. The distribution of flow throughput within the two extrema is not reflected in the fairness value.

A second metric called the *equality fairness* metric is introduced to assess the throughput distribution. The fairness metric developed by Jain et al. [JCH84] is applied to the normalized throughput of the flows of a protocol.

$$F_P^{\text{equality}} = \frac{\left(\sum_{f \in P} B_f\right)^2}{|P| \sum_{f \in P} {B_f}^2} \tag{4.8}$$

The metric is dimensionless, independent of scale, and is bounded by $1/|P|$ and one. Again, a fairness value of one is reached for an equal resource distribution. A value of $1/|P|$ results when one flow claims all the resources and other flows are starved.

4.1.4 Metrics for throughput variation

Variability

The *variability* metric is used to analyze how often and to what extent a protocol changes its sending rate. The lower the variability of a flow, the more resilient the protocol is to noise. The variability is measured by comparing the flow throughput for different granularities. At any point of time, the variability metric $G_{f,(\delta_1,\delta_2)}(t)$ is given by

$$G_{f,(\delta_1,\delta_2)}(t) = \frac{|B_{f,\delta_1}(t) - B_{f,\delta_2}(t)|}{B_{f,\delta_2}(t)} \tag{4.9}$$

for $\delta_1 < \delta_2$. Comparison of the variability of flows over time allows to assess responsiveness of the flows and their susceptibility to noise. The average variability over the measurement interval $[t_0, T]$ is accordingly defined as

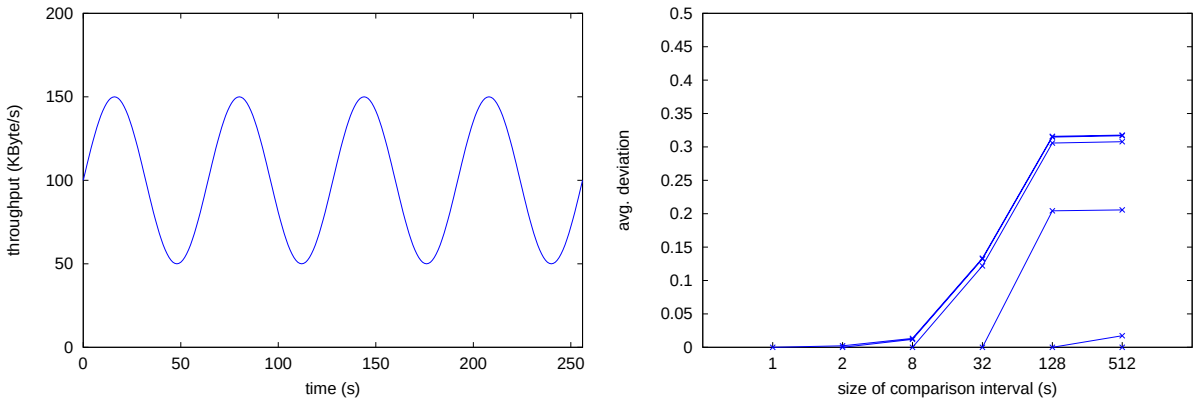
$$G_{f,(\delta_1,\delta_2)} = \sum_{i=1}^{(T-t_0)/\delta_1} \frac{G_{\delta_1,\delta_2}(t_0 + \delta_1 i)}{T/\delta_1} \tag{4.10}$$

The absolute value of $G_{f,(\delta_1,\delta_2)}$ varies with the amplitude of the oscillations, while the ratio of δ_1 to δ_2 reveals throughput stability over different timescales and thus relates to the oscillation frequency. A flow with a constant throughput has a variability of zero. Perfect oscillation between a throughput of zero and $2B_f$ results in a variability of one. The maximum variability is reached, when the throughput is zero for all intervals apart from one peak value.

To demonstrate the use of the average variability metric, figure 4.1 shows the average variability metric applied on a sending rate that oscillates with a frequency of 1/64Hz (i.e. a complete increase/decrease cycle every 64 seconds). Each line in figure 4.1b) represents the variability for a fixed value of δ_1 and for different δ_2 values. Lines start with a deviation of zero where $\delta_1 = \delta_2$. The larger the difference of short-term

throughput and long-term throughput, the higher the variability measure for the corresponding δ_1/δ_2 combination. $G_{f,(\delta_1,\delta_2)}$ values for δ_1 values below 32 seconds are very similar, which suggests that there are no short-term oscillations present in the signal. Likewise, the similarity of the 128 second curve and the 512 second value indicate absence of oscillations on a timescale above 128 seconds. Since maximum and minimum throughput of the signal are 32 seconds apart, the 32 second curve differs from the 8 second curve and the 128 curve. Its proximity to the 8 second curve indicates, that the oscillation frequency is closer to 32 seconds than to 128 seconds.

The information obtained from the average variability measure is comparable to applying discrete fourier transform (DFT) on the rate signal. While it represents the frequency spectrum very coarsely and single frequencies are hard to identify, the variability measure is well suited to capture the overall stability of the signal. It does not suffer from aliasing and the so-called *picket-fence effect* of the DFT.



a) Frequency 1/64Hz

b) Variation measure

Figure 4.1: Variability measure for different sending rates

Coefficient of Variation

Another measure to assess the variability of the throughput is the coefficient of variation (CoV). It is defined as the standard deviation of a time series over the mean of the time series.

$$CoV_{f,\delta} = \frac{\sqrt{\frac{1}{(T-t_0)/\delta}\sum_{i=1}^{(T-t_0)/\delta}\left(B_{f,\delta}(t_0+\delta i)-B_f\right)^2}}{B_f} \tag{4.11}$$

The coefficient of variation is dimensionless and independent of scale. A high CoV indicates high variability of the time series.

4.1.5 Responsiveness

Responsiveness corresponds to the speed with which the protocol can adapt to changed network conditions. Assume that a network parameter $v(t)$, such as RTT, loss rate, or available bandwidth changes at a point of

time t'. Responsiveness is defined as the amount of time Δ_t it takes the TFRC protocol to adjust its estimate of the parameter $p_v(t)$ to the "true" value with a maximum deviation of 10%.

$$R_{p_v}(v) = \min\left\{\Delta_t : 0.9\, v(t') \leq p_v(t' + \Delta_t) \leq 1.1\, v(t')\,,\ \Delta_t \geq 0\right\} \tag{4.12}$$

This metric is difficult to apply in uncontrolled network environments, where often only the estimate of a parameter but not its real value is known. Moreover, it is necessary that once the parameter changed at time t' it stays constant for at least the response time $R_{p_v}(v)$.

4.2 Test environments

To capture protocol behavior over a wide range of network conditions, the protocol was tested on the Internet as well as on local area networks (LANs), modem connections and artificially created network conditions.

The protocol performance is closely related to the packet size. To be able to compare results for TFRC and TCP, it is important to ensure that the size of the TFRC packets matches the TCP packet size.[2]

4.2.1 Real-world experiments

Since the receiver already gathers all important variables for the TCP-model, protocol internal data is logged at the receiver for protocol evaluation. Thus, variables measured at the receiver are up-to-date, while the RTT samples computed at the sender are logged with a delay of $1/2$ RTT.

We use *tcpdump* [JLM89] to monitor incoming and outgoing packets independent of the receiver log files. In particular, this facilitates a sanity check of the protocol, as differences between the reported sending rate and the number of transmitted packets can easily be identified.

4.2.2 Dummynet

In the Internet, it is not possible to study protocol behavior under well defined conditions. While the protocol design itself was done by means of a network simulator [MF99], the implementation for "real" networks has to be verified against the simulated protocol.

Dummynet [Riz97] allows to simulate arbitrary network conditions. It is implemented as a filter in the protocol stack of the operating system. Packets that are handed from one network layer to another can be intercepted and passed through so-called *pipes*. These pipes can be modified to simulate bandwidth limitations, buffer size, propagation delay, and packet drop rate. While it is essential to study certain aspects of the protocol under controlled conditions, the results cannot directly be used to infer protocol behavior in the Internet. Due to the nature of pipes, no random cross-traffic is present.[3] The packet filtering in the pipes causes a delay that is not similar to the packet handling delay caused by routers. However, some network

[2] Common packet sizes are 576-1500 bytes [MD90].

[3] A makeshift way to simulate cross-traffic is to modify the packet drop rate and thus create random loss not related to a buffer overflow. However, this crude method can by no means capture the complexity of Internet cross-traffic.

conditions are very rare in the Internet and the use of tools like Dummynet is inevitable to study protocol behavior under such conditions.

The propagation delay mentioned in conjunction with Dummynet experiments is *not* the RTT. The RTT consists of propagation delay and buffer delay. The buffer delay depends on the buffer utilization. Likewise, the specified packet drop rate is *additional* to packet loss that occurs due to buffer overflow. It is used to simulate random loss due to a high level of statistical multiplexing or lossy network links (e.g. wireless connections).

4.3 Steady state protocol behavior

To better understand the results of the real-world experiments in the following sections, it is helpful to analyze the behavior of TFRC (and TCP for reasons of comparison) in steady state. Steady state analysis is only possible in highly controlled environments but not in dynamic environments such as the Internet. For this reason, we set up a *Dummynet* link between two hosts on the same subnetwork. This makes specific modifications to the network conditions possible. The experiments were conducted with one flow at a time to preclude that competing flows affect the protocol behavior. This makes a study of isolated effects of network parameters on the protocol possible.

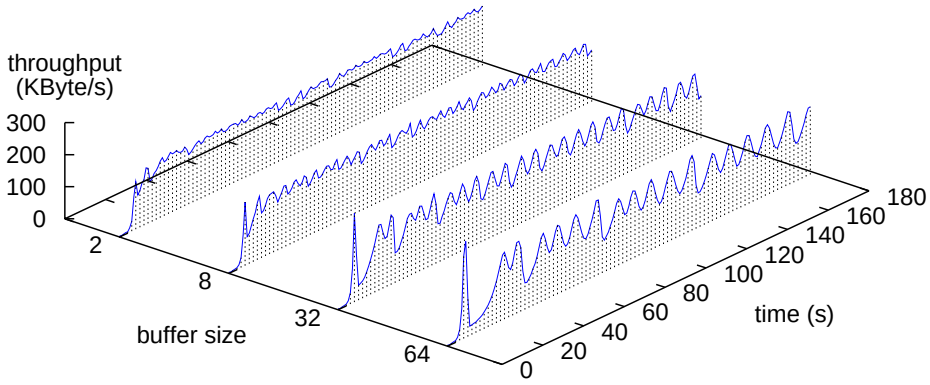
Figures 4.2 to 4.4 show the results of the experiments. To facilitate the comparison of the protocol behavior with different parameters, the two-dimensional throughput graphs of the experiment series are combined into one single three-dimensional graph. The z-axis shows the protocol throughput, the y-axis is the time-axis, and the parameter under investigation is shown on the x-axis. Each experiment series is done for TFRC without ISM[4], TFRC with ISM, and TCP. Without competing flows, the impact of ISM on the protocol behavior is quite large because the RTT depends almost solely on the sending rate (via the buffer delay).

4.3.1 Impact of buffering

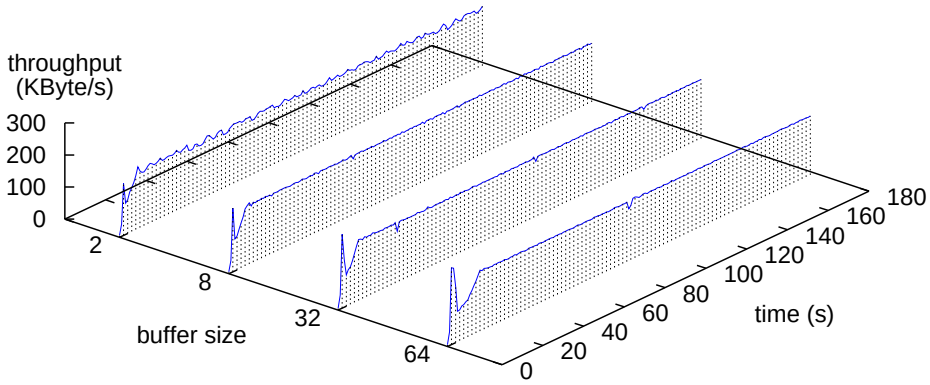
The first experiments studied the impact of the amount of buffering on a TFRC flow. The setup for the experiments was a fixed propagation delay of 100 ms and a bottleneck bandwidth of 1.5 MBit/s. No additional packet drop rate was specified. The average RTT was around 160 ms, leading to a bandwidth-delay product of 30 KByte, or roughly 20 packets with a size of 1460 bytes. The buffer size varied between 2 and 64 packets, thus ranging from virtually no buffering at all to high overbuffering.

The results of the buffer experiments are given in figure 4.2. The graph of TFRC without ISM shows the typical sending rate oscillations of TFRC on an empty link. As long as the buffer is not full, the protocol experiences no loss and increases its rate. The larger the buffer size, the longer the connection can sustain a packet rate that is higher than the link bandwith. Since the sending rate still increases while the buffer fills up, a larger buffer size causes a higher sending rate when the buffer flows over. Therefore, more packets have to be dropped. As a consequence, not only the buffer delay but also the loss event rate vary more. Only a completely underbuffered connection (e.g. with a buffer size of $1/10$ of the bandwith-delay product) does not show these increase/decrease cycles.

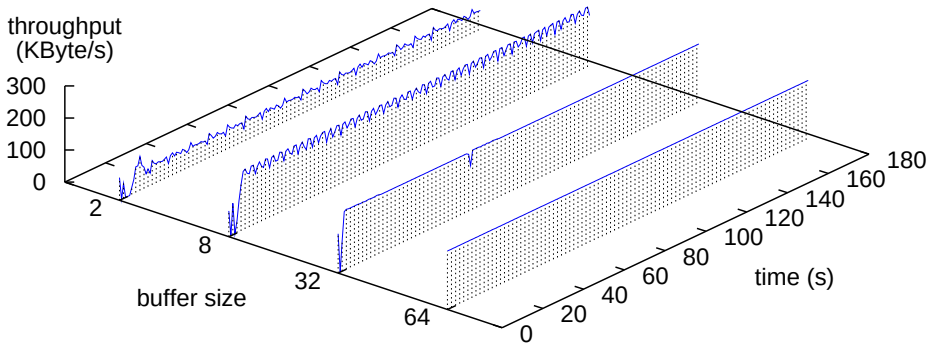
[4] Inter-packet Space Modulation is described in 3.6.4



a) TFRC without ISM



b) TFRC with ISM



c) TCP

Figure 4.2: Impact of buffering on the sending rate

The oscillations can be prevented by modulating the inter-packet spacing. Highly underbuffered connections still show small variations in the sending rate caused by variations in the high loss rate. At the same time, all variations caused by the relationship of RTT and buffer delay are removed completely. Apart from one "dent" in the sending rate after slowstart, when history faking causes a reduction of the sending rate by half, TFRC shows a steady sending rate at exactly the link bandwidth.

While TFRC's performance does not suffer when the link is underbuffered (a lower RTT makes up for a higher loss event rate), TCP does very badly under such conditions. The buffer is not provisioned for the number of outstanding packets TCP needs to reach equilibrium. With a buffer size of 2 packets, TCP achieves a throughput of only 30% of the available bandwidth, while TFRC utilizes nearly 100%. On a mildly underbuffered link and without competing flows, TCP only has a small performance penalty and achieves a throughput close to the link bandwidth, as packet loss is always indicated by triple duplicate ACKs and not by timeouts. TCP does not lose its ACK-clocking. When TCP has sufficient buffer space, its self-clocking capabilities result in a sending rate at exactly the link bandwidth. No packet loss occurs on the link and the buffer level corresponds to the bandwidth-delay product. Under such conditions TCP outperforms TFRC, although both have the same throughput. While ISM balances out variations in RTT and loss event rate, it does not prevent loss events. TFRC experiences periodic loss events with a loss frequency dependent on the buffer size. Also the buffer level is not stable as with TCP but varies in relation to the sending rate.

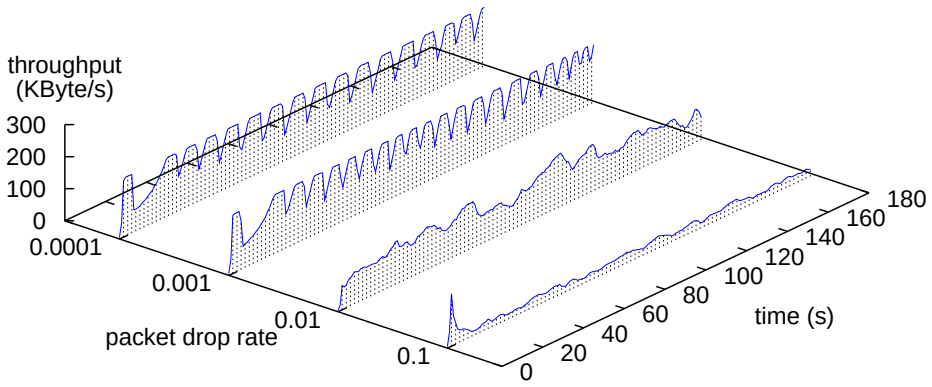
4.3.2 Impact of packet loss due to lossy links

In the experiments in this section we study the impact of packet loss not caused by a buffer overflow but faulty transmission on protocol performance. While such loss is rare in "conventional" networks, it is common on satellite connections or other forms of wireless communication. These conditions can be simulated in Dummynet by specifiying a probability with which data packets are dropped. This loss probability varied between 0.01% and 10%. The buffer size was set to the bandwidth-delay product, the link speed to 1.5 MBit/s, and the propagation delay to 100 ms.

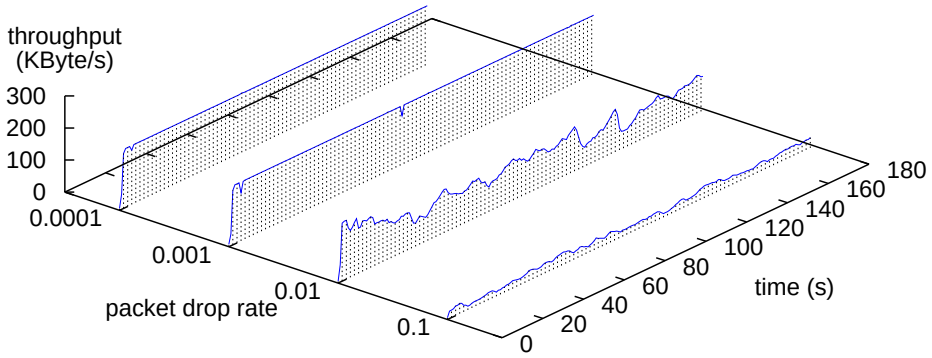
Figure 4.3a) shows, that the TFRC sending rate oscillations fade with an increasing packet drop rate. At high packet drop rates, TFRC experiences a loss and thus reduces its sending rate before the buffer has time to fill up. Under such conditions, ISM is not effective and the sending rate of TFRC with and without ISM is similar. The average throughput of TFRC is 100 KByte/s with a drop rate of 1% and 20 KByte/s with a 10% drop rate. This corresponds to the throughput given by the complex TCP equation for a loss event rate of 1% and 10% respectively and an RTT of 100 ms.[5]

TCP achieves a throughput of roughly 90 KByte/s for a 1% packet drop rate. However at a 10% drop rate, TCP's throughput is reduced to merely 10 KByte/s, half of the TFRC throughput. Under such high loss rates, TCP is not able to maintain its self-clocking and experiences timeouts frequently. As noted in section 2.2.2, the complex model is adjusted to Sack TCP which performs better than other TCP variants under such conditions. The TCP implementation that was used with Dummynet is a variant of Reno TCP which explains the different performance.

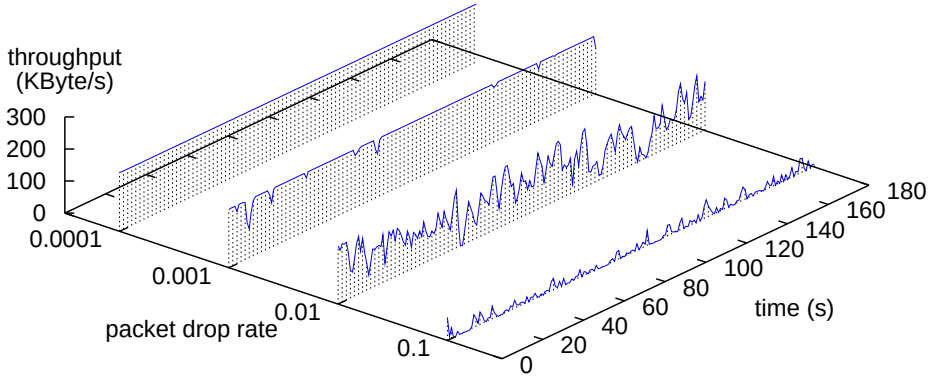
[5] Note that the average TFRC throughput of an experiment is not always equal to the throughput computed by the TCP equation using the average loss rate and RTT of the experiment. This is mainly due to rate increase/decrease limits of the protocol and smoothing of the parameter estimates.



a) TFRC without ISM



b) TFRC with ISM



c) TCP

Figure 4.3: Impact of packet loss due to lossy links

4.3.3 Impact of propagation delay

The last of the experiment series investigated the impact of propagation delay on the protocol performance. The setup resembles the setup for the previous experiments. Here, the propagation delay varied between 1 ms and 250 ms.

Since the RTT consists of propagation delay and buffer delay, different propagation delays influence the frequency of the sending rate oscillations in TFRC without ISM. The smaller the overall RTT, the higher the oscillation frequency, as can be seen in figure 4.4a). Again, these oscillations are removed when ISM is used. In TFRC as well as TCP, the maximum rate increase depends on the RTT. Thus, with a higher RTT it takes both protocols longer to reach the link bandwidth, once they get out of slowstart. Apart from that, TFRC (with ISM) and TCP achieve a throughput of the link bandwidth, independent of the RTT.[6]

4.4 Inter-protocol fairness

How the TFRC protocol interacts with competing TCP flows in real networks is a major concern, since TCP-friendliness is one of the main design goals of the protocol. To evaluate the TCP-friendliness, we examined several combinations of TFRC and TCP flows using the inter-protocol fairness metric F_{TFRC}^{inter}.

The total number of experiments is too high to present every result in detail. This section gives an overview of aggregated fairness per TCP/TFRC combination. In section 4.7, some representative experiments are discussed in more detail.

4.4.1 Transcontinental connections

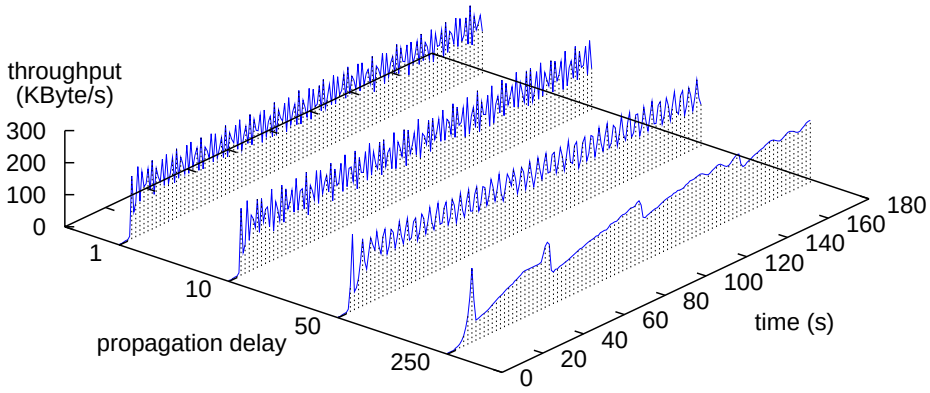
A test suite consists of 4 consecutive test runs of one specific TCP/TFRC combination. This provides resilience against exceptional network conditions that might flaw a test run. The results of the test runs of one suite are averaged. We assume that no fundamental permanent changes in the network conditions occur during a test suite, especially the route from sender to receiver should stay the same. Since the timescale for route changes in the Internet is usually several days [Pax97b], this is a reasonable assumption.

A total of four test suites was run for each combination of flows on different days of the week and at different daytimes. The total duration of each of the test runs was 180 seconds. Different startup characteristics such as initial sending rate and the amount of time for which the protocol stays in slowstart mode influence the sending rate for some amount of time after the tests are started. To prevent the results from being affected by this startup behavior, the first 60 seconds of each experiment were excluded from the analysis.

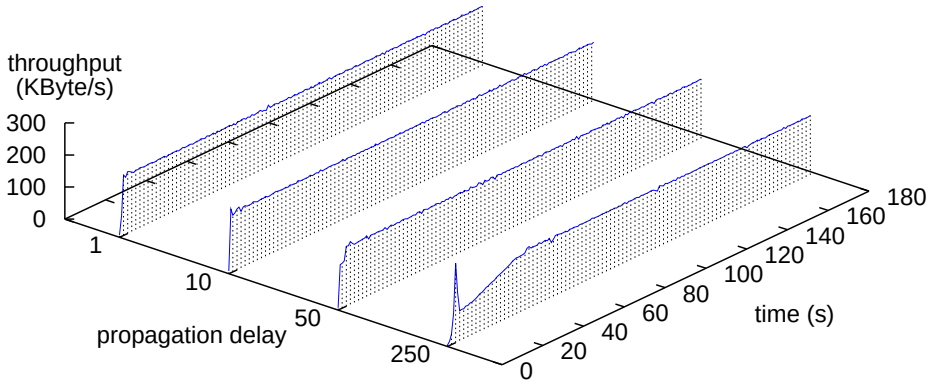
While receivers were started on a machine connected to UC Berkeley's campus network, the senders were situated on distant networks in Europe.[7] The overall network conditions for the test runs are given in table 4.1. It lists average RTT, average packet drop rate, and the total bandwidth utilization of all flows.

[6] If the propagation delay would be increased to such an extent that the bandwidth-delay product exceeded the size of the buffer, the TCP connection would again not be able to achieve maximum throughput, while TFRC performance would be basically unaffected.

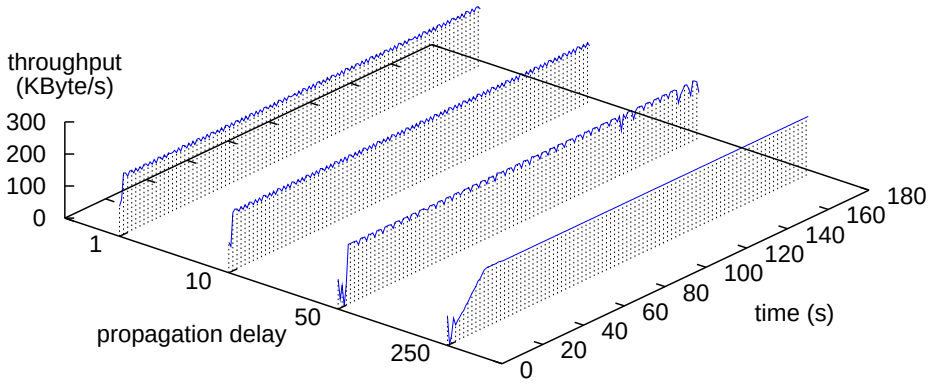
[7] Validation experiments where sender and receiver locations are exchanged led to the same test results.



a) TFRC without ISM



b) TFRC with ISM



c) TCP

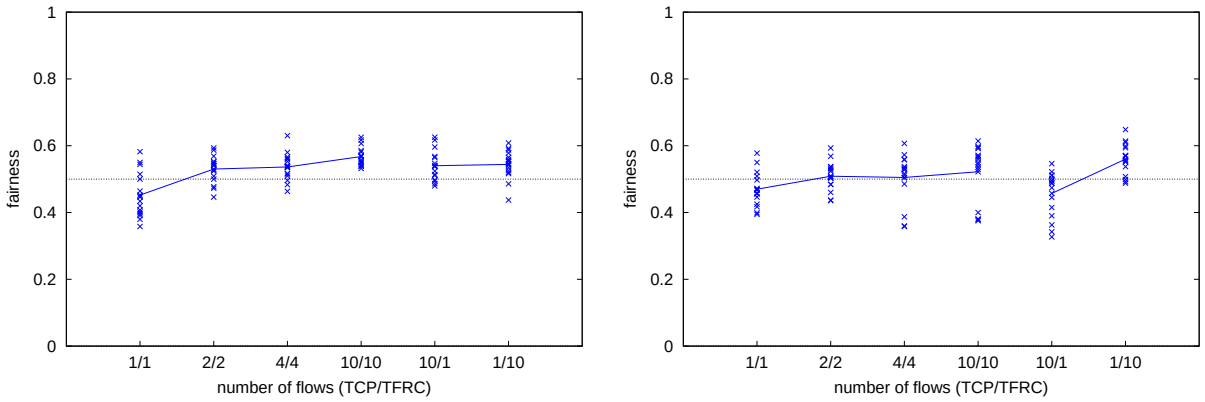
Figure 4.4: Impact of propagation delay

Both transcontinental connections show similar characteristics. The average RTT is around 250 ms, rising only moderately with the number of connections, while the average loss rate of a flow increases in proportion to the number of simultaneous connections. Thus, the bandwidth share of a single flow is mostly determined by its loss rate. The total bandwidth utilization increases moderately in proportion to the number of flows.

Host location	**TCP flows**	**TFRC flows**	**avg. RTT**	**avg. loss**	**band-width**
UCL	1	1	209 ms	0.0544%	536603
	2	2	240 ms	0.1739%	546687
	4	4	248 ms	0.4604%	564033
	10	10	279 ms	1.4095%	609807
	10	1	266 ms	0.8290%	536222
	1	10	251 ms	0.4884%	692236
UMANN	1	1	207 ms	0.0588%	437049
	2	2	223 ms	0.1594%	487852
	4	4	241 ms	0.3641%	552042
	10	10	272 ms	1.3152%	577261
	10	1	252 ms	0.6060%	453185
	1	10	251 ms	0.3927%	702935

Table 4.1: Network conditions (UCL/UMANN)

The aggregated results of the tests are shown in figure 4.5. A measurement point in the graphs corresponds to the average fairness value of all the flows of one test run with a specific TCP/TFRC combination. These fairness values are again averaged per TCP/TFRC combination and connected with line segments to show how fairness changes in relation to the number of competing flows.



a) UCL b) UMANN

Figure 4.5: Aggregated inter-protocol fairness (UCL/UMANN)

Experiments with the sender at the University College London, GB (UCL) show increasing fairness with the number of flows. While the same holds true for the experiments from a host at the University of Mannheim,

GER (UMANN), there the fairness is more related to the number of TFRC flows than to the total number of flows. This can be seen by the different fairness values for the experiments with 1 TFRC and 10 TCP flows compared to the experiments with 10 TFRC and 1 TCP flows. The variation in fairness values is larger in the experiments with UMANN, which correnponds to higher variations in the network conditions. All in all, both connections show a very good fairness behavior of TFRC.

4.4.2 Inner-US connections

The setup for protocol tests to a host located at the University of Massachusetts (UMASS) in the US was similar to the transcontinental setup. Again, 16 test runs with a duration of 180 seconds were carried out for each of the TFRC/TCP combinations. The network conditions for the experiments are shown in table 4.2.

Host location	TCP flows	TFRC flows	avg. RTT	avg. loss	band-width
UMASS	1	1	89 ms	2.0882%	126718
	2	2	95 ms	1.8217%	244137
	4	4	111 ms	1.8324%	447950
	10	10	153 ms	2.4535%	557524
	10	1	126 ms	2.3891%	360108
	1	10	121 ms	2.0269%	564237

Table 4.2: Network conditions (UMASS)

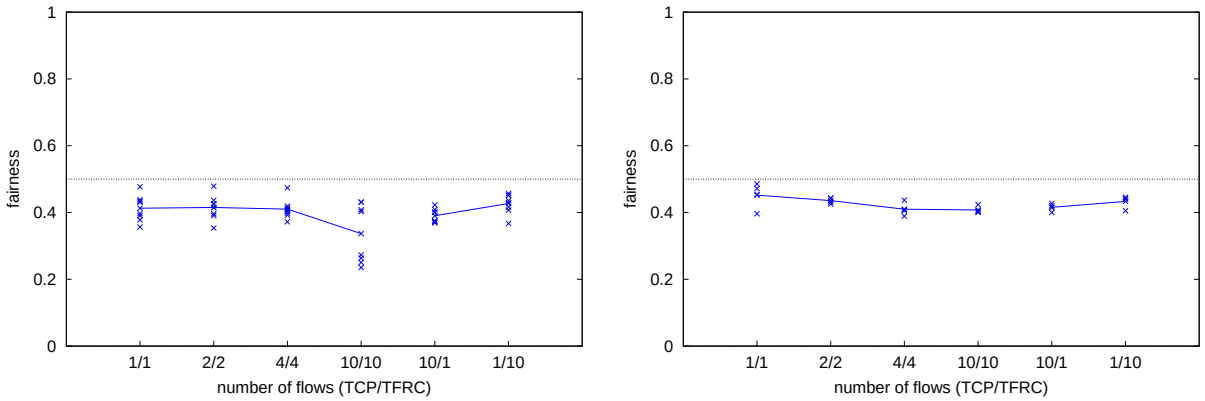
UMASS has a high speed access to the Internet, shared by many users. As in the transcontinental connections, the RTT increases moderately with the number of flows. However, here the loss rate remains fairly constant on a high level. This results in an increase of the total bandwidth utilization roughly in proportion to the number of flows.[8]

The average fairness results for the tests from UMASS in figure 4.6a) reveal, that TCP on average achieves a throughput of 40% of the overall throughput. Timeouts occur frequently in the experiments with UMASS due to the high loss rate. Analyzation of the TCP tracefiles revealed a surprisingly high number of duplicate packets at the receiver. Duplicate packets results from packet duplication in the network or a very aggressive timeout value. It turned out that the latter was the case for the experiments, thus hurting TCP's performance. RTT variations (in particular with many competing flows) caused the retransmission timer to expire, resulting in unnecessary retransmissions. The operating system on the host at UMASS is Solaris 2.7, known for a rather small timeout value.

Experiments with a different host using Linux as the operating system show better TCP performance.[9] The different TCP implementation of Linux with a higher timeout value achieves a higher throughput than Solaris TCP. This results in an increased overall fairness level and in particular prevents the decrease in fairness with a larger number of flows.

[8] In such an environment, *applications* can compete unfairly with other applications by using more than one connection at once for the data transfer.

[9] TFRC performance is similar in the experiments with the Solaris and the Linux host.



a) Solaris b) Linux

Figure 4.6: Aggregated inter-protocol fairness (UMASS)

4.4.3 Environments with a low level of statistical multiplexing

In this section, we discuss the results of test runs over a cable modem connection and a T1 line. As mentioned before, the model assumptions of independent RTT and loss rate do not hold in environments with a low level of statistical multplexing. It is necessary to also analyze protocol behavior under unfavorable conditions.

Dedicated T1 line

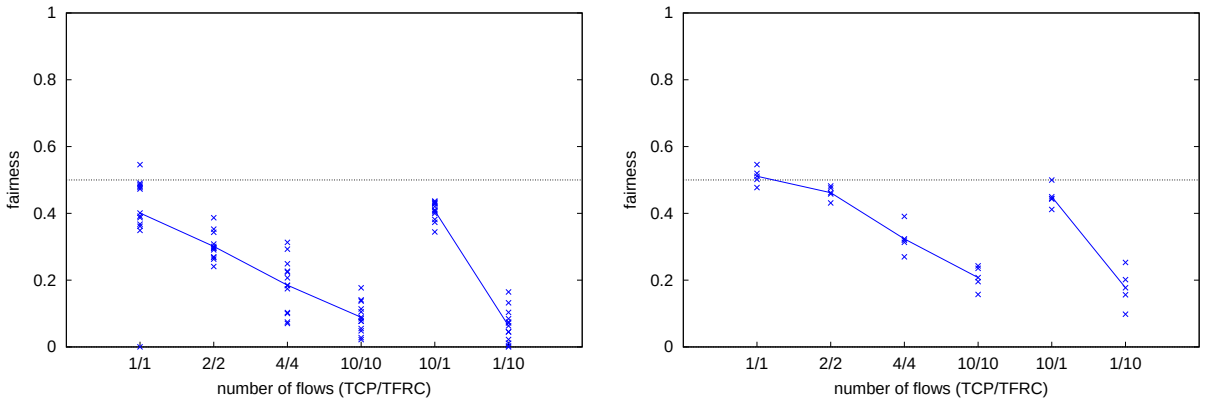
Nokia, Boston, is connected to the Internet over a 1.5 MBit/s T1 line. The experiments were conducted during non-working hours and virtually no cross-traffic was present on the Nokia LAN. On the path to Nokia, congestion occurs mainly at the border-router between Nokia and the Internet due to the lack of cross-traffic. The buffer level was solely determined by the TCP and TFRC sending rate. The overall bandwidth utilization remains constant at roughly 180 KByte/s, the maximum available bandwidth of a T1 line. Unlike in the previous experiments where the RTT increased moderately, here the RTT remains nearly constant. The buffer delay is not influenced by the number of flows because the buffer is fully utilized most of the time. Buffer space is scarce at the border-router to Nokia.

Host location	TCP flows	TFRC flows	avg. RTT	avg. loss	band-width
NOKIA	1	1	169 ms	0.5234%	178163
	2	2	175 ms	1.2058%	180792
	4	4	182 ms	2.4739%	183367
	10	10	189 ms	5.5091%	185824
	10	1	181 ms	3.9983%	184674
	1	10	203 ms	4.2023%	183822

Table 4.3: Network conditions (Nokia, Boston)

There is a fundamental difference in the behavior of the TFRC and of the TCP protocol. TCP depends on

a certain number of outstanding packets in the network to be able to reach full performance. If the buffers are not provisioned for that number of packets, TCP performance suffers, while TFRC's performance is not dependent on the amount of buffering. Furthermore, there are some situations where TCP sends packets back-to-back (i.e. without inter-packet space). This increases the likelihood of packet drops when there is only little buffer space. TFRC on the other hand always leaves an inter-packet gap. When the next packet has to be sent, it is likely that another packet has been removed from the buffer in the meantime, thus making space for the current packet.



a) TFRC without ISM b) TFRC with ISM

Figure 4.7: Aggregated inter-protocol fairness (Nokia, Boston)

Figure 4.7a) shows very low TFRC fairness for a higher numbers of flows. With an increasing number of concurrent flows, the loss rate for each flow has to increase, since the available bandwidth is fixed. The likelihood of TCP timeouts increases in an environment with a high loss probability. When a TCP connection experiences a timeout, it does not send packets for a short period of time. The TFRC connections see a lower loss event rate and RTT and consequently increase their sending rate and fill up the buffer. With the increased sending rate and buffer level, it is harder for TCP to get back to its fair share of bandwidth. After some time, there is not enough buffer space left for TCP to successfully do slowstart after a timeout. It suffers from its own burstiness. When TCP experiences consecutive timeout events, the time interval before TCP attempts slowstart again increases exponentially with the number of tries. Because of the scarce buffer space, TCP flows experienced exponential back off to up to 48 seconds during the tests runs. TCP backs of by such an amount only under very high loss rates well above 20%. At the same time, the TFRC connections experienced an average loss rate of only 5%. As shown in figure 4.7a), TCP achieves a throughput of only a fraction of TFRC's throughput for a high number of concurrent flows.

Behavior improves considerably when TFRC uses ISM which causes variations in the inter-packet spacing. This increases the burstiness of TFRC to some extent. It is easier for TCP to compete against bursty flows in such an environment. The available buffer space varies more and TCP has a fair chance to perform a successful slowstart after a timeout. While TFRC still achieves a higher throughput than TCP, the pathological behavior where TCP had little or no throughput at all with a large number of flows is avoided.

When the direction of the connection is reversed (not shown in figure 4.7), with the sender at the local host and the receiver at Nokia, the conditions for TCP improve further. The path from the sender to the

congested router is not just the LAN, but a multi-hop Internet connection across the US. The present cross-traffic generates noise in the RTT signal and congestion can occur everywhere along the path. While the border-router at Nokia is still the main bottleneck for the connection, the path through the Internet introduces additional noise, further increasing the variations in the buffer level.

Modem connection

This section describes experiments with TCP and TFRC over a cable-modem connection. Up-stream bandwidth of the cable modem is limited to 128 KBit/s, which explains the overall bandwidth utilization of up to 12 KByte/s.

TCP flows	TFRC flows	avg. RTT	avg. loss	band-width
1	1	2216 ms	0.5312%	11738
2	2	2687 ms	0.9029%	11049
4	4	3138 ms	2.2439%	10278
10	10	3621 ms	3.5942%	10578
10	1	3202 ms	2.9570%	9567
1	10	3322 ms	2.4338%	8728

Table 4.4: Network conditions (cable modem)

The cable modem connection uses a very large buffer. While the propagation delay is around 80 ms, the RTT samples in the experiments vary from 100 ms to up to 4000 ms. A large buffer favors TCP connections. Whenever TCP slowstarts, it is likely that it can reach a relatively high bandwidth because there is sufficient buffer space for several back-to-back packets. Although the network conditions are similar to the Nokia connection as far as cross-traffic is concerned, TFRC has a lower throughput than TCP in most cases. For larger numbers of flows, fairness is near optimal. This is remarkable for the fact that the granularity of the sending rate is very low. The overall throughput is less than ten packets per second or around twenty packets per average RTT. Thus, in the ten TCP and ten TFRC flow tests, each flow has to send on average one packet per RTT for optimal fairness. Obviously, the flows can only send no, one or several packets per RTT, so they send at either the optimal rate, 100% below, or 100% (or more) above.

4.4.4 Dummynet experiments

Since Dummynet connections are not subject to cross-traffic but provide a rather static environment, only 4 experiments were run per TFRC/TCP combination. As before, the experiment duration was set to 180 seconds. The bandwidth range comprised two orders of magnitude and varied from the equivalent of a slow modem connection (28.8 KBit/s) to a T1 line (1.5 MBit/s), as shown in table 4.5. Since the link was shared only by the TFRC and TCP connections of the experiment, their sending rate has a large influence on the buffer delay and the loss rate. To decouple sending rate, RTT and loss rate to some extent, an additional delay of 50 to 200 ms and an additional packet drop rate of 1.0% to 2.0% was used. Thus, loss events occur not only when the buffer overflows and the RTT is not solely dependent on the buffer level. The average network conditions encountered during the Dummynet experiments with the afore mentioned setup are shown in table 4.6.

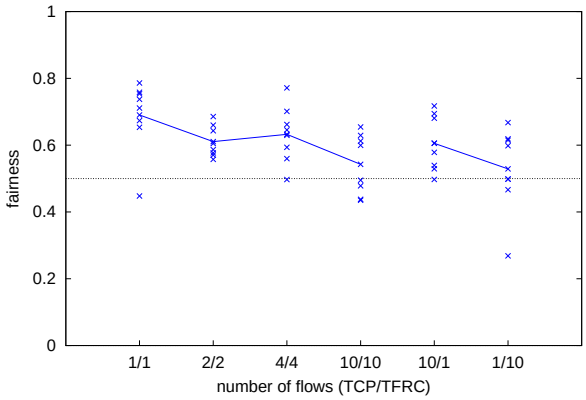


Figure 4.8: Aggregated inter-protocol fairness (cable modem)

The number of packets per RTT increases in proportion to the available bandwidth. A high number of packets allows for more fine-grained rate changes. While decrease in fairness with an increasing number of flows is present in all the Dummynet experiments, the absolute value of the fairness metric is higher for high bandwidth experiments because of the increased rate granularity. The fairness values for all of the Dummynet experiments are shown in figure 4.9.

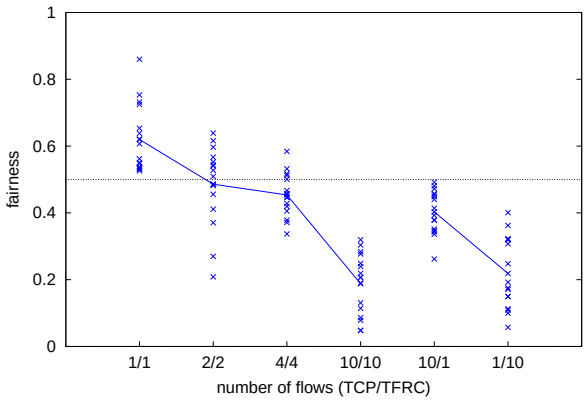


Figure 4.9: Aggregated inter-protocol fairness (Dummynet)

4.5 Intra-Protocol Fairness

TFRC intra-protocol fairness is measured using the two metrics defined in section 4.1.3. Experiments were carried out with 2, 4, 10, and 20 concurrent TFRC flows and no competing TCP flows. To be able to compare TFRC's and TCP's intra-protocol fairness, the same experiments were run solely with TCP flows. Test results are shown separately for experiments with a large amount of cross-traffic and a low amount of cross-traffic. The former experiments comprise 16 tests each with UCL, UMASS, and UMANN, while the latter experiments comprise the tests with Nokia and the cable modem connection.

In environments with a high level of statistical multiplexing, both TCP and TFRC show a very high *equality*

bandwidth	delay	buffer size	drop rate
1500 KBit/s	50 ms	20 pkts.	1.00%
384 KBit/s	100 ms	20 pkts.	1.00%
128 KBit/s	200 ms	20 pkts.	1.00%
28.8 KBit/s	200 ms	40 pkts.	2.00%

Table 4.5: Dummynet setup

TCP flows	TFRC flows	avg. RTT	avg. loss	band-width
1	1	186 ms	1.1573%	101489
2	2	240 ms	1.9036%	104680
4	4	273 ms	3.8466%	109663
10	10	284 ms	6.0587%	118410
10	1	290 ms	4.3195%	108791
1	10	287 ms	5.0695%	113140

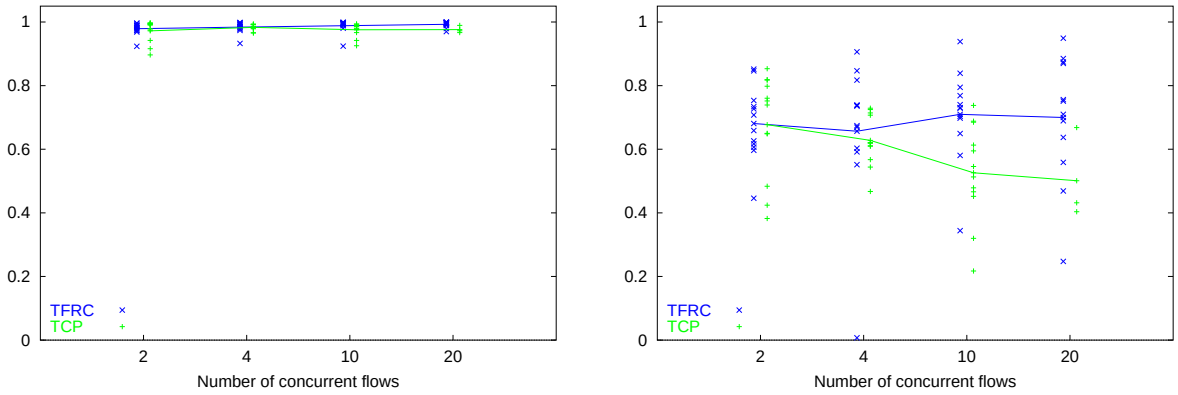
Table 4.6: Network conditions (Dummynet)

fairness (figure 4.10a))[10]. The intra-fairness of TFRC is only marginally higher than the fairness of TCP. The *max-min* fairness values are lower than the equality fairness values, since max-min fairness only covers the extreme cases of throughput. TFRC shows good max-min fairness results (figure 4.10b)). The flow with the lowest throughput still achieves on average of 75% of the flow with the highest throughput. TFRC's max-min fairness is independent of the number of flows. TCP's max-min fairness is in the same range for a small number of concurrent flows, but decreases to about 55% for a larger number of flows.

Intra-fairness decreases drastically in environments with a low level of statistical multiplexing, as depicted in figure 4.11b). Both, TFRC's and TCP's inter fairness values are inversely proportional to the numbers of flows. While the fairness values for just two competing flows resemble the fairness values of the experiments with a high level of statistical multiplexing, fairness deteriorates considerably with ten flows or more. Especially the max-min fairness shows, that the range of TCP throughput is very large and some flows are even starved completely. TFRC can retain a higher intra-fairness level under such circumstances and starvation does not occur.

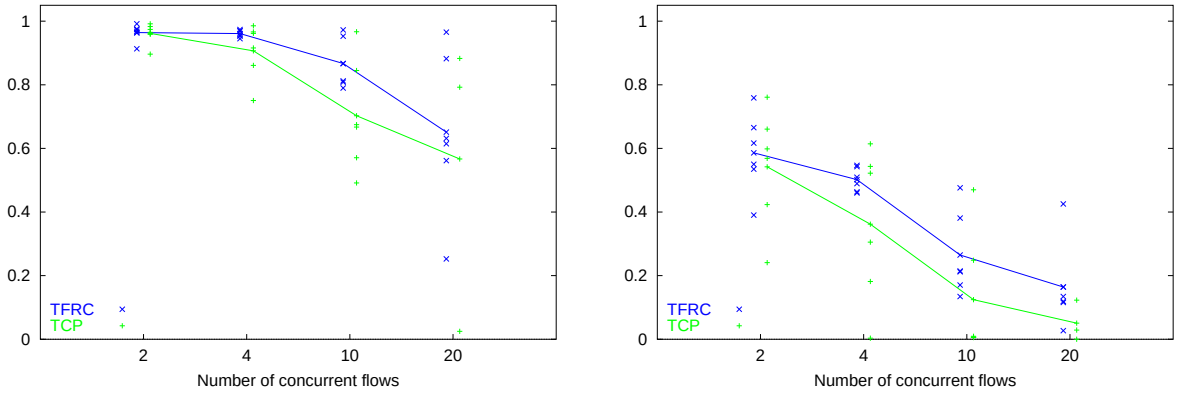
In all test environments, TFRC shows better intra-protocol fairness than TCP. On average, TFRC's intra fairness is 20% higher compared to TCP's intra-protocol fairness.

[10]TFRC and TCP data points are slightly shifted to more clearly distinguish data points.



a) equality b) max-min

Figure 4.10: High level of statistical multiplexing



a) equality b) max-min

Figure 4.11: Low level of statistical multiplexing

4.6 Throughput variation / Stability

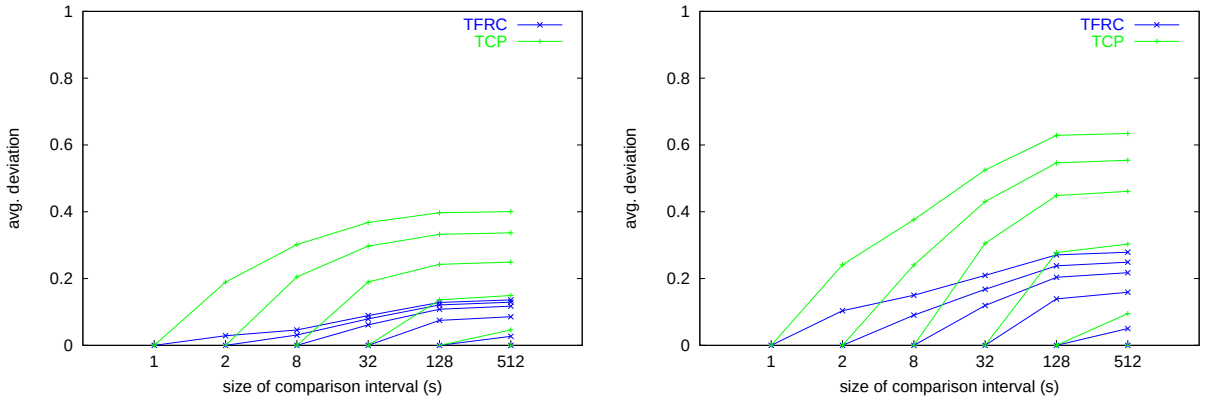
4.6.1 Variability

The variability measurements are based on the data of the inter-protocol fairness experiments. For a specific δ_1/δ_2 combination, the variability metric $G_{f,(\delta_1,\delta_2)}$ was computed for each TFRC and TCP flow and the results were averaged over the number of flows. Each averaged variability value represents one data point in figure 4.12. Again, experiments were divided into tests with a low level of statistical multiplexing (Nokia, cable modem) and tests with a high level of statistical multiplexing (UCL, UMANN, UMASS). The δ_1/δ_2 combinations are shown in table 4.7.

For all the experiments and all the δ combinations, TFRC has a lower variability than TCP. This was to be expected from the steady state behavior of TCP and TFRC as discussed in section 3.4.2. In figure 4.12a),

δ_1 in seconds	δ_2 in seconds
1	1, 2, 8, 32, 128, 512
2	2, 8, 32, 128, 512
8	8, 32, 128, 512
32	32, 128, 512
128	128, 512
512	512

Table 4.7: δ_1/δ_2 combinations



a) High level of statistical multiplexing

b) Low level of statistical multiplexing

Figure 4.12: Variability

the steepest ascend in variability for TCP is for low δ_2 values. This indicates large differences in short-term TCP throughput. Long-term throughput of TCP on a timescale of more than 32 seconds is rather stable.

The proximity of the 1-, 2-, and 8-second δ_1 curves for the TFRC flows indicate a very stable throughput on short timescales. High granularity throughput compared to throughput for δ_2 values of 128 and 512 also reveals no significant deviation. Thus, most of the changes in throughput are on a timescale of 32 to 128 seconds. Absolute variability values are always lower for TFRC than for TCP.

The results for environments with a low level of statistical multiplexing show comparable behavior. The absolute throughput deviation is larger, but throughput changes happen on the same timescales. Thus, figure 4.12b) resembles a scaled version of figure 4.12a).

4.6.2 Coefficient of Variation of the throughput

Analyzing the throughput variability for the Internet experiments by means of the coefficient of variation corroborates the findings of the previous section. Figure 4.13 depicts the coefficient of variation over different timescales. The left part of the figure shows results for TFRC, the right part shows results for TCP. All CoV values decrease for larger timescales, as the granularity of the CoV measurement gets closer to the

duration of the experiment.

TFRC throughput varies generally less and in particular has better stability characteristics on small timescales below a few seconds. The transcontinental experiments with UCL and UMANN show the lowest variation in TFRC as well as TCP throughput because of a high level of statistical multiplexing. Connections to UMASS on the other hand have a very high CoV caused by high variations in the loss rate. The graph shows very nicely that unfairness in the UMASS experiments with the Solaris host is the result of poor TCP behavior. Because of many unneccesary timeouts and subsequent slowstarts, the CoV for TCP is much higher than in all the other experiments, while the TFRC CoV of that experiment is in the same range as the other TFRC CoVs.

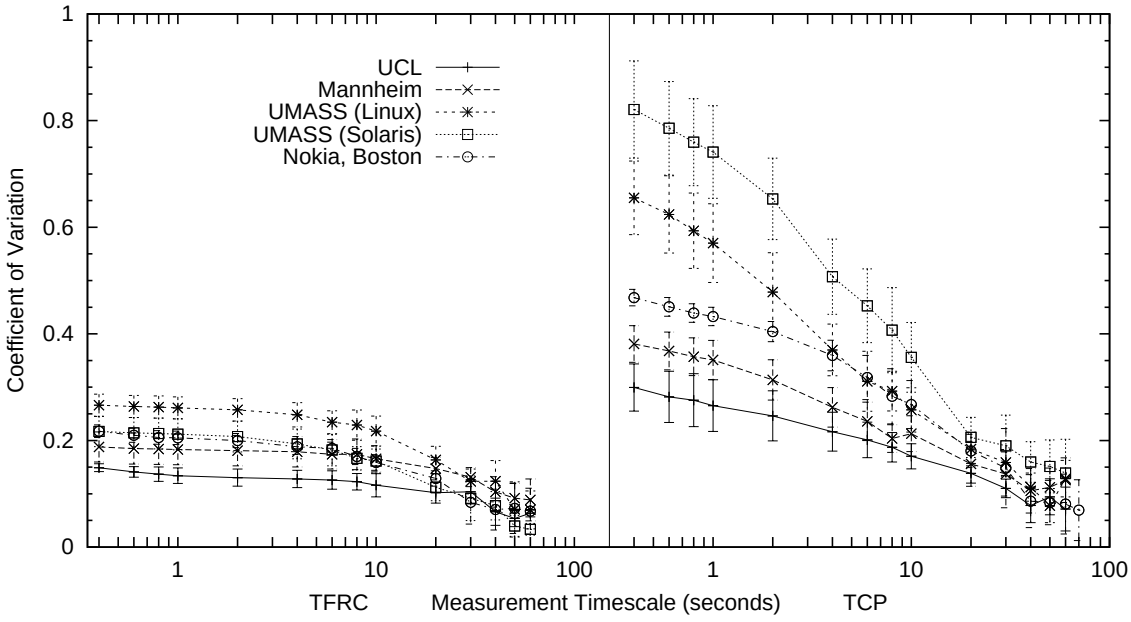


Figure 4.13: Covariance of throughput

4.7 Selected experiments in detail

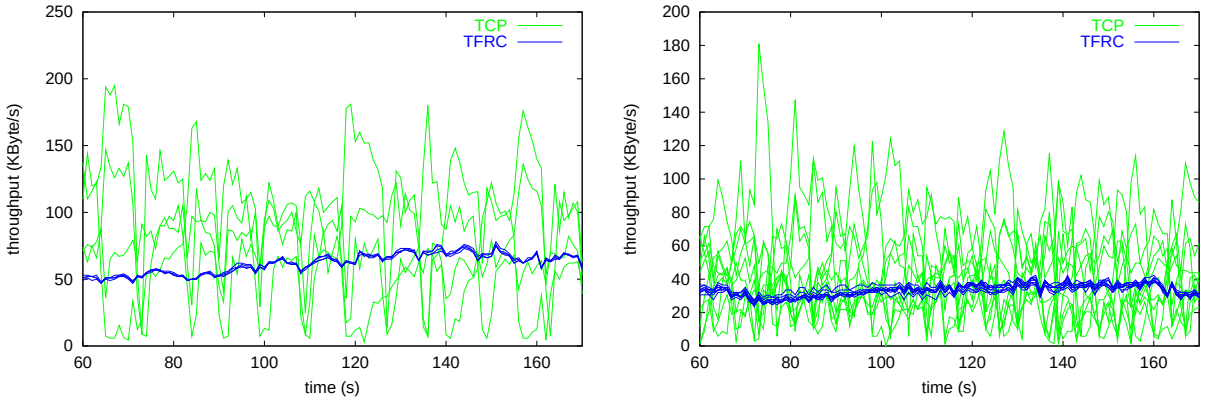
In this section we present a few of the over 800 protocol experiments. Some of the test runs are analyzed in detail to show how TFRC and TCP flows perform over time.

4.7.1 University College London

Of the UCL experiments, we show one test run with four TCP and four TFRC flows and one test run with ten flows for each protocol. Conditions on the link to UCL are very steady and tests runs with different numbers of flows show very similar behavior.

The sharp drops in TCP throughput in both figure 4.14a) and 4.14b) correspond to TCP timeouts. TFRC shows near optimal fairness with a share of 44% of the overall throughput in the experiment with four flows

each and 45% in the experiment with ten flows. Thus, TFRC behaves slightly too conservative. Max-min intra-protocol fairness of TFRC was very high with a value of 0.97 and 0.94 respectively, indicating that the throughput of the flows was nearly identical. TCP had lower inter-protocol fairness values of 0.72 and 0.61. Thus, the slowest TCP flow achieved only a throughput of 61% of the fastest TCP flow.



a) 4 TCP and 4 TFRC flows

b) 10 TCP and 10 TFRC flows

Figure 4.14: UCL experiments

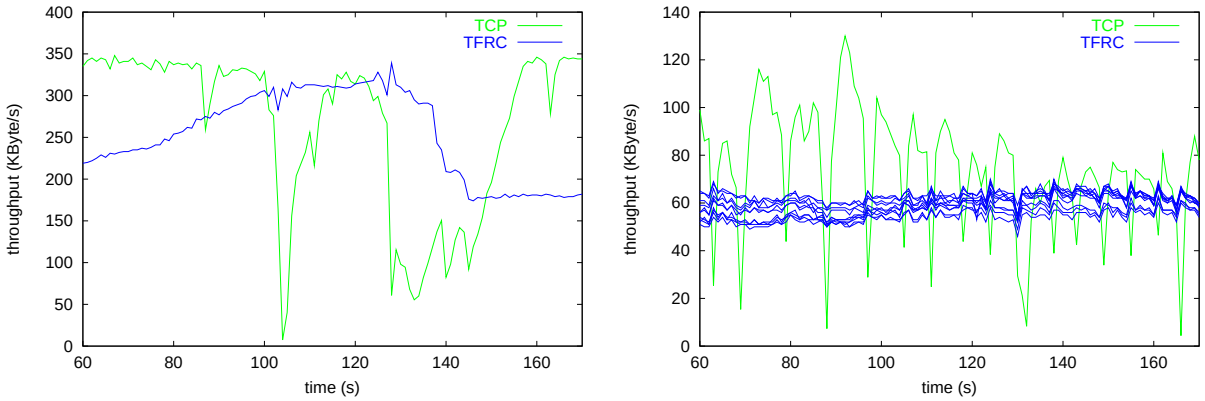
4.7.2 University of Mannheim

Most of the experiments to UMANN are similar to the corresponding experiments to UCL, since both connections go over the atlantic link. The experiment with one TFRC and one TCP flow emphasizes the different responsiveness of the two protocols. Although TFRC's fairness value in this experiment was 0.51 (i.e. both flows achieve the about same throughput), the throughput of TCP and TFRC over short timescales is quite different. Shortly after 100 seconds, there is substantial congestion in the network which lasts for a few seconds. The TCP connection has a timeout, while the TFRC connection adjusts its throughput only marginally. The congestion dissipates before TFRC reacts to it. More persistent network congestion which lasts for 20 seconds occurs at a time of 125 seconds. Again, TCP reacts almost immediately due to a timeout, while it takes TFRC around 15 seconds to reduce its rate substantially. For TFRC to be fair over long timescales, it has to increase the rate slowlier than TCP becasue of its lower responsiveness to congestion. After the congestion, TCP needs 10 seconds to increase its rate back to the original value, while TFRC needs much more time until the loss history adjusts to the lower loss event rate.

Figure 4.15b) shows ten TFRC flows competing against one TCP flow. As in the experiments with UCL, TFRC has a high inter-protocol fairness as well as intra-protocol fairness value. TCP encounters periodic timeouts roughly every ten seconds, while TFRC maintains a very stable sending rate under the same conditions.

4.7.3 University of Massachusetts

TCP achieved slightly higher throughput than TFRC in the transatlantic experiments. This is not the case in the experiments to UMASS, partly because of the relatively high loss rate, partly because of the broken

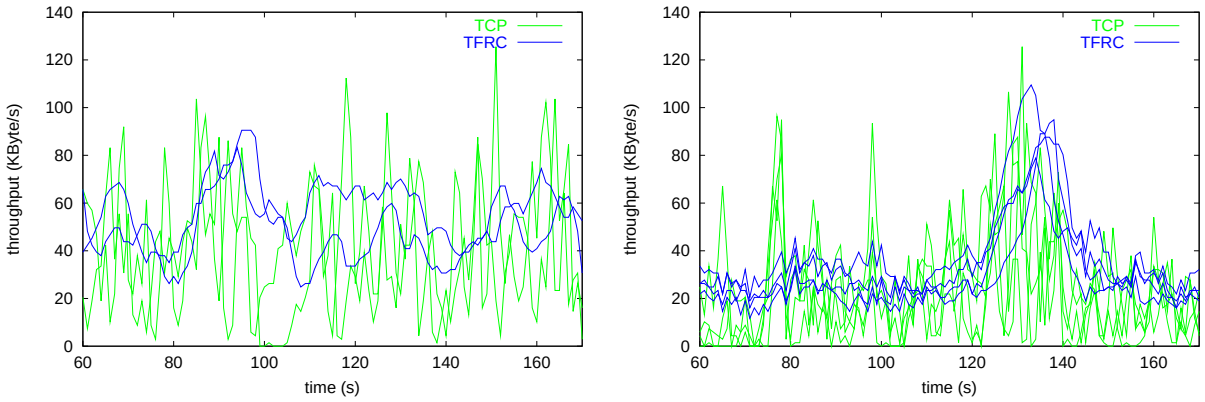


a) 1 TCP and 1 TFRC flow

b) 1 TCP and 10 TFRC flows

Figure 4.15: UMANN experiments

TCP implementation on the Solaris host. TFRC's fairness values are close to optimal although TFRC throughput slightly exceeds TCP throughput in almost all of the experiments. In the experiment depicted in figure 4.16a), there is a sharp drop in available bandwith at a simulation time of 100 seconds. While TCP immediately experiences a timeout and reduces its sending rate to zero, TFRC adapts more slowly and adjusts to TCP throughput only after ten seconds. When changes in the bandwidth are more gradual, TCP and TFRC behave very similar. In the experiment depicted in figure 4.16b) there is a temporary increase in available bandwidth from second 120 to second 140. Increase and decrease of the throughput of the flows takes place simultaneously and to the same extent for TCP and TFRC.



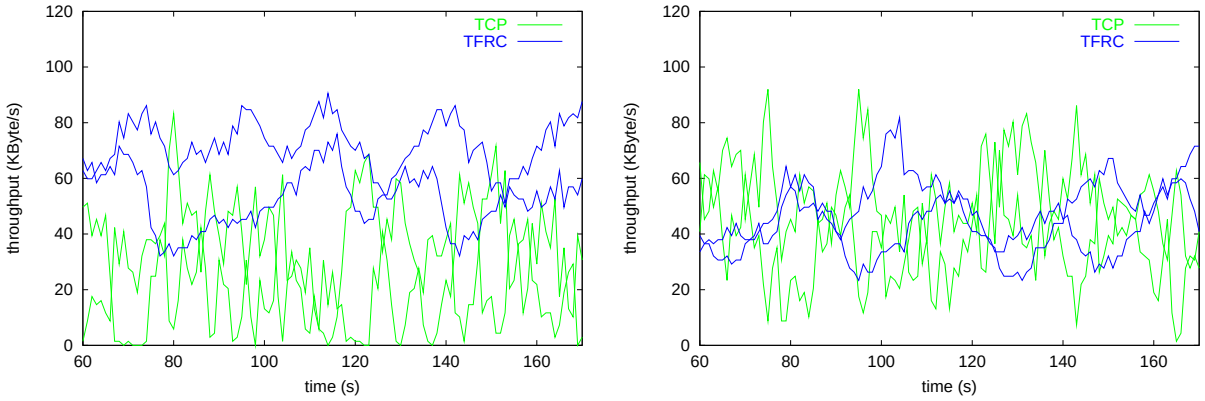
a) 2 TCP and 2 TFRC flows

b) 4 TCP and 4 TFRC flows

Figure 4.16: UMASS experiments

4.7.4 Nokia, Boston

Figures 4.17a) and 4.17b) show the fairness improvement in the experiments with Nokia by using ISM. Timeouts occur frequently in both experiments, but when TFRC uses ISM it takes TCP less time to get back up to speed. The cumulative time each TCP flow spent waiting for the retransmission of a packet during the whole experiment decreased from about 50 seconds to 18 seconds. In addition, more packet losses were indicated by triple duplicate ACKs instead of timeouts.



a) 2 TCP and 2 TFRC flows without ISM

b) 2 TCP and 2 TFRC flows with ISM

Figure 4.17: Nokia experiments

4.7.5 Long-term experiments

The previous experiments for inter- and intra-protocol fairness all have a duration of three minutes. To compare long-term throughput of TCP and TFRC, we conducted additional experiments with a duration of half an hour. Tests were done with two TFRC and two TCP flows. Figure 4.18 shows the average throughput over time. Throughput was measured with a granularity of 20 seconds to remove the effects of short-term rate changes from the graph and show long-term behavior.

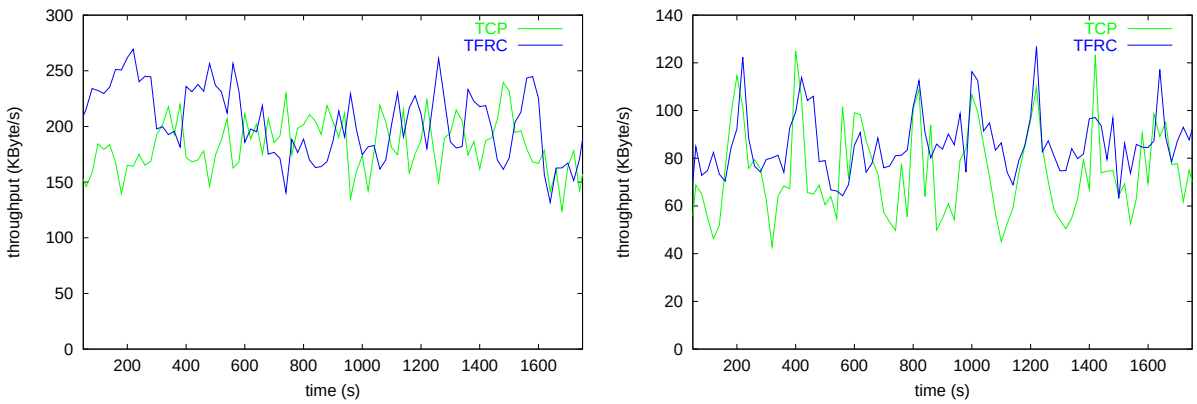
The previous three minute experiments revealed, that TFRC competes fairly with TCP under most network conditions and show less variability on short timescales. These long-term experiments show that the behavior of TFRC and TCP is very similar over long timescales.

4.8 Responsiveness

The TFRC protocol reacts to changes more gradually than TCP. In this seciton we investigate, on what timescales TFRC adjusts to changed network conditions and if TFRC's lower responsiveness can harm other flows.

The following scenarios were used to assess responsiveness:

- changes in the RTT,



a) UCL b) UMASS

Figure 4.18: Long-term protocol behavior

- changes in the loss rate,
- competing flows that start or stop data transmission, and
- variations in the available bandwidth.

The RTT responsiveness of TFRC was only simulated and not measured in experiments to preclude interference of the buffer delay with the RTT adaption time. The responsiveness of the loss estimate was simulated in respect to the different deweighting methods, since the goal of history deweighting is an improved responsiveness to a decrease in the packet drop rate. Dummynet experiments were conducted to investigate TFRC's behavior when competing flows start or stop sending and when the available bandwidth varies.

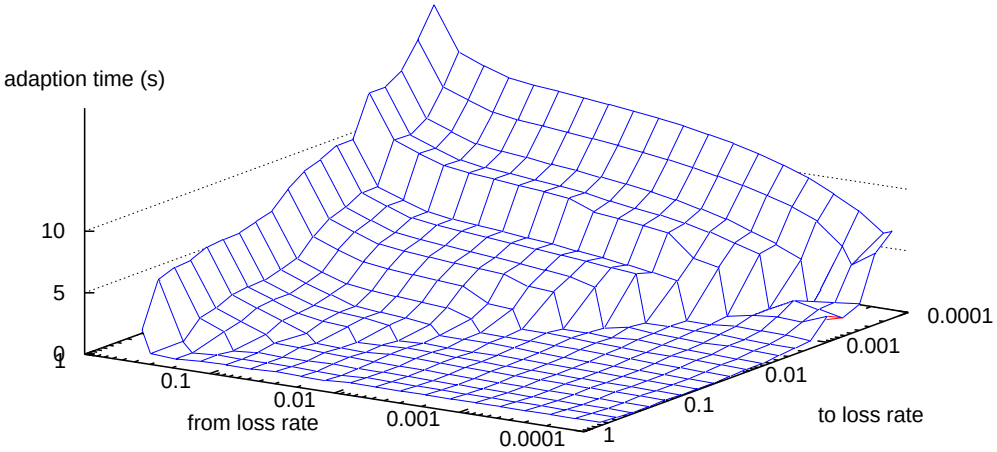
4.8.1 Simulations

Impact of different deweighting methods on the responsiveness of the loss estimate

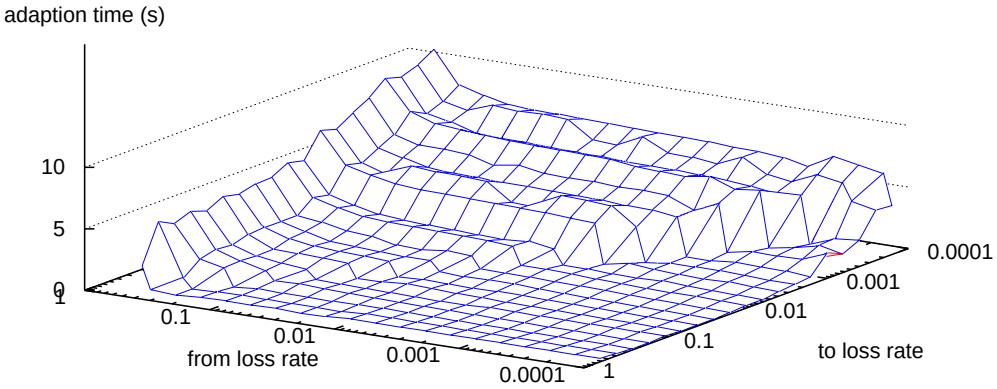
To assess the impact of deweighting on the loss estimate, we simulated how long it takes the different methods to adapt to changes in the loss rate. To isolate the effect of deweighting, the RTT was fixed at 50 ms and loss events were independent of the sending rate. Responsiveness was measured as the time interval until the loss estimate lies within a 10% interval of the "real" value. We conducted the simulations with periodic loss events as well as random loss events and for different loss history sizes n. For random loss events, the results were averaged over a number of simulations.[11]

The main reason for low responsiveness is a change from a high to a low loss rate. The receiver gets few new loss events and old loss intervals age very slowly and affect the estimated loss rate for a long time. Another reason for low responsiveness is a low sending rate caused by a high loss rate. Since only few packets arrive per time interval, it takes a long time until the receiver obtains enough packets to have a reasonable estimate of the new loss rate.

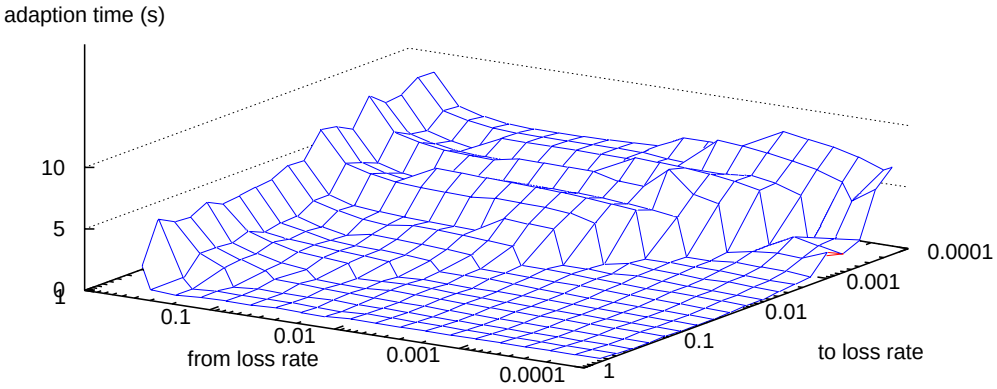
[11] For reasons of brevity, figure 4.19 only presents simulation results for $n = 8$ and loss events with a Poisson distribution.



a) No deweighting



b) Loss interval duplication



c) Proportional deweighting with loss history remodeling

Figure 4.19: Alternatives for additional deweighting

Loss interval duplication (figure 4.19b)) and proportional deweighting with a lower bound d_{lim} of 0.2 (not shown) perform roughly similar and provide improved responsivenes over loss estimation without deweighting (figure 4.19a)). Proportional deweighting without a lower bound on the deweighting factor but with remodeling of the loss history (figure 4.19c)) shows the highest responsiveness, in particular, when the change in the loss rate is over several orders of magnitude.

Simulations with a higher number of loss intervals and with a constant weight distribution show reduced responsiveness.

Responsiveness to RTT changes

Responsiveness to RTT changes was simulated in a similar manner by abstracting from the impact of the sending rate on the RTT. Let r_1 be the initial value of the smoothed RTT t_{RTT}. All new RTT samples indicate a change in RTT to a value r_2. Figure 4.20 shows the time it takes TFRC until the smoothed RTT lies within a 10% interval of the "target" value (i.e. $0.9r_2 \leq t_{RTT} \leq 1.1r_2$). In the simulation, receiver reports with a new RTT sample arrive exactly once per RTT without any delay jitter.

Responsiveness is mainly dependent on the decay factor of the EWMA for the smoothed RTT. Responsiveness was measured for a smoothed RTT with a decay factor of 0.01 and a decay factor of 0.05. The smaller the decay factor the longer the adaption time. A decay factor of 0.05 is a reasonable compromise between stability of the RTT estimate and responsiveness. Responsiveness is high when the absolute difference of the old and the new RTT samples $|r_1 - r_2|$ is small. The protocol also shows good responsiveness for a low absolute value of the new RTT samples. A low RTT results in a high frequency of receiver report and thus a fast adaption to changes.

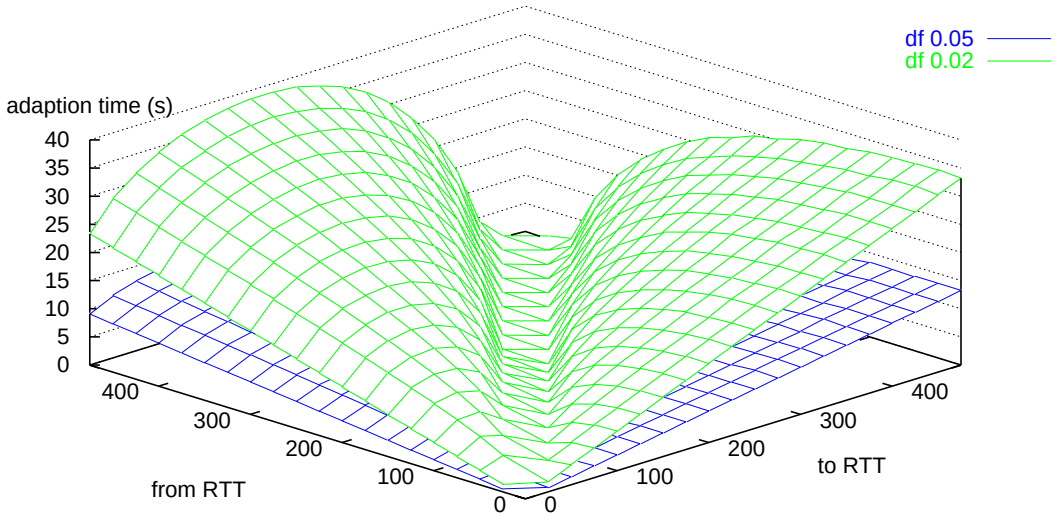


Figure 4.20: Responsiveness to RTT changes

4.8.2 Responsiveness to an increase and decrease in the number of flows

Responsiveness to an increase and decrease in the number of flows was investigated in Dummynet. Several setups with different on-off times of the flows were used.

The total duration of the experiments was 240 seconds. An 80 second interval between changes of the conditions allowed the system to settle again. The link bandwidth was set 1.5 MBit/s and the propagation delay was set to 100 ms. An additional packet drop rate of 0.5% provided the necessary variations in the buffer level as a substitue for cross-traffic. The exact on-off times of the flows a shown in figure 4.21.

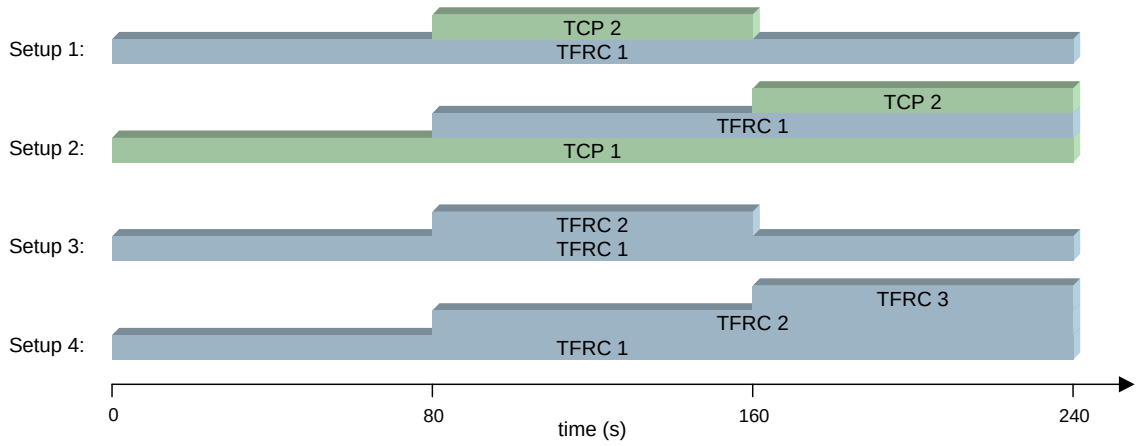
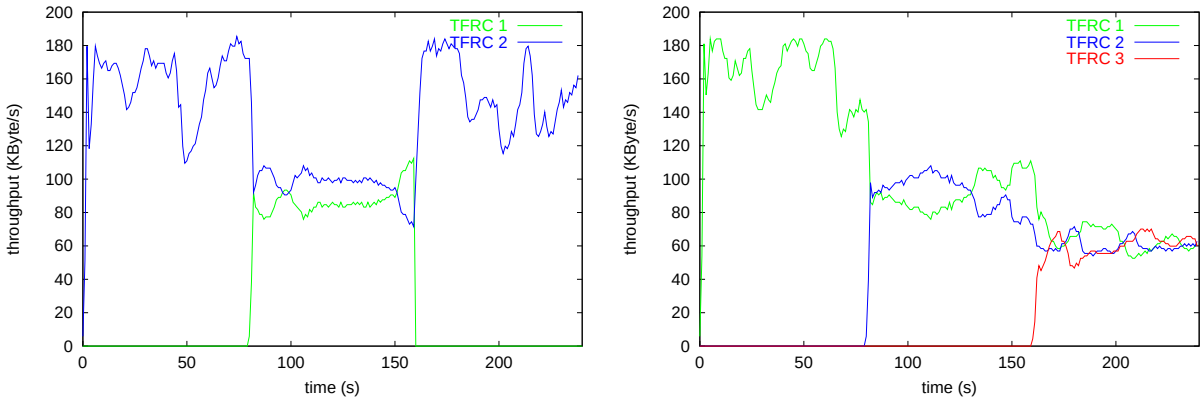


Figure 4.21: Experiment setup

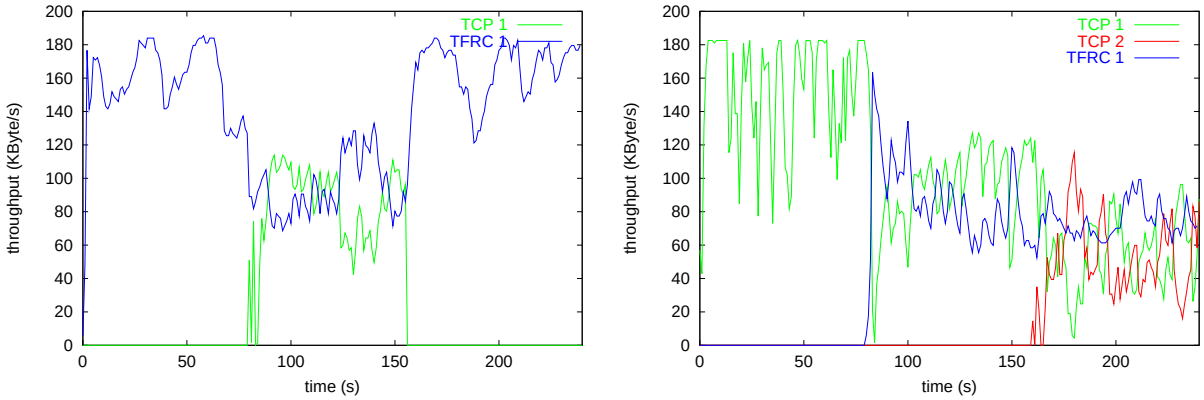
In the experiments with *Setup 1*, TFRC exhibits very good responsiveness to an increase and decrease in the number of competing flows (figure 4.22). The experiments show flow throughput, not the sending rate as reported by the sender. Slowstart of the second TFRC flow ends at about twice the fair share of the flows bandwidth, but the peak in the packet rate is dampened by buffering and packet drops. The receiver sees an increase to almost exactly the fair share of bandwidth. When the second TFRC flow stops sending, the first flow gets back to its original bandwith in a matter of a few seconds. The same holds for *Setup 2*, where three TFRC flows are started with a spacing of 80 seconds. Flows level off at their fair share of bandwidth very fast.

Figure 4.23 shows the responsiveness with a mix of TCP and TFRC flows. Responsiveness is still good, but the burstiness of TCP causes more variations in the flow throughput. In the experiment with *Setup 3*, it takes TCP several slowstart attempts until TFRC backs off enough so that TCP can reach its fair share of bandwidth. TFRC exceeds its fair share of bandwidth during slowstart, when only a TCP flow is present in *Setup 4*. After about 20 seconds, the TCP and TFRC flows settle. The start up of a second TCP flow at time 160 seconds reduces the throughput of the first TCP flow, while TFRC adjusts its rate slowly. Only towards the end of the experiment does TFRC back off enough so that both TCP flows can reach their fair share of bandwith. Average TCP throughput from second 160 on is 70% of TFRC throughput.



a) TFRC impulse b) TFRC startup behavior

Figure 4.22: Experiments only with TFRC flows



a) TCP impulse b) TCP/TFRC startup behavior

Figure 4.23: Experiments only with TCP and TFRC flows

Responsiveness to changes in the available bandwidth

The responsiveness experiments in the previous section investigated how a constant amount of bandwith is distributed among a varying number of flows. In this section, responsiveness to changes in the available bandwidth itself is examined. The experiment duration was 180 seconds. Table 4.8 shows how available bandwidth varied over time. *Setup 1* features a 40 second reduction of the available bandwith by 50%. *Setup 2* simulates two dropouts during the experiments where the bandwidth drops to zero. The dropouts have a duration of 10 and 30 seconds respectively. In contrast to the previous experiments, figure 4.24 shows the sending rate of the flows (without packet drops), not the throughput.

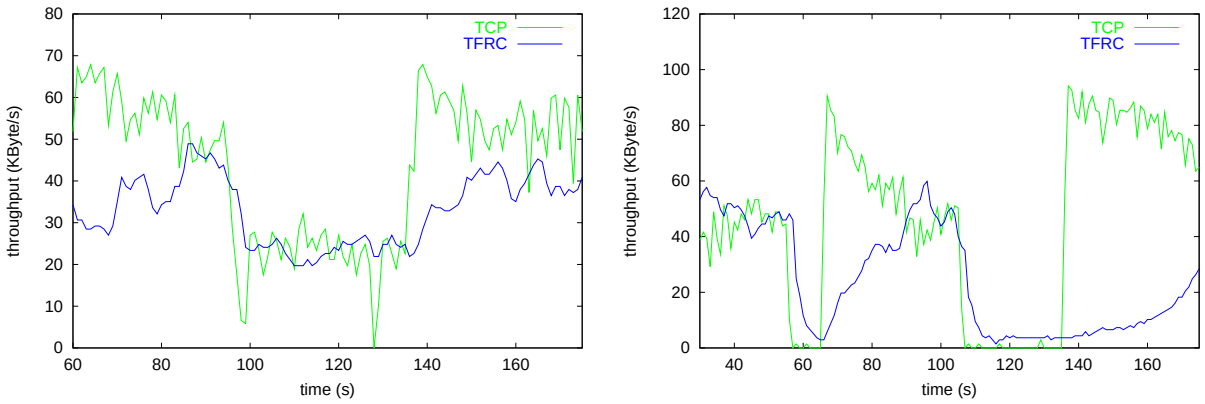
As can be seen from figure 4.24a), TCP immediately reacts to the reduced bandwidth with a timeout. TFRC decreases its sending rate slowlier and TCP and TFRC reach their fair share of bandwidth 5 seconds after the bandwidth reduction. When the bandwidth limitation is suspended after 40 seconds, TCP almost im-

Setup 1	
second 0 - 80	1500 KBit/s
second 80 - 120	750 KBit/s
second 120 - 180	1500 KBit/s

Setup 2	
second 0 - 60	1500 KBit/s
second 60 - 70	0 KBit/s
second 70 - 110	1500 KBit/s
second 110 - 140	0 KBit/s
second 140 - 180	1500 KBit/s

Table 4.8: Experiment setup

mediately consumes the additionally available bandwith. TFRC responds over a longer timescale and only after 20 seconds the equilibrium between TFRC and TCP that existet before the bandwidth reduction is restored. TFRC's rate increase was even more slow in the experiments with connection dropouts. Again, TCP increases and decreases its rate immediately after the conditions change. After a 10 second dropout, TFRC needs the same amount of time as in the bandwidth halving experiment to get back up to speed. However, after a 30 second dropout, TFRC does not get back up to its fair share of bandwidth during the remainder of the experiment. It only achieves about a third of TCP sending rate because the aging of the loss history takes time.



a) Responsiveness to halving of the available bandwidth

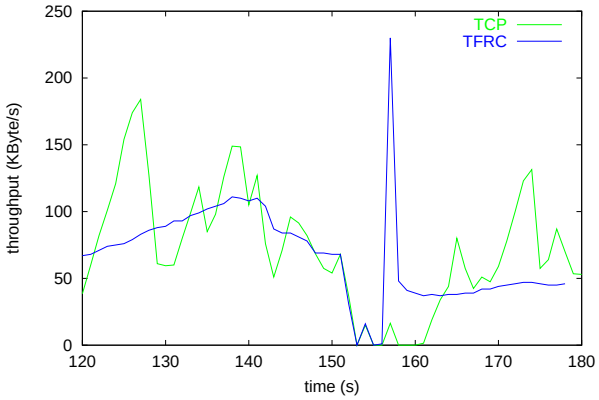
b) Responsiveness to drop-outs with different durations

Figure 4.24: Responsiveness to changes in the available bandwidth

4.9 Response to dropouts

During tests with UCL, some connections experienced dropouts for up to three seconds as shown in figure 4.25a). During these specific dropouts, packets were merely delayed and only few were lost. This explains the different response of TFRC and TCP as opposed to the dropout experiments in Dummynet. Under the present conditions, the TCP connection times out and performs slowstart as soon as packets start to flow again. The TCP sending rate is very low after the dropout while the TFRC protocol reduces its sending rate by a smaller amount. After the dropout, there is a short burst of packets, caused by the simultaneous delivery of the delayed packets. Because no receiver reports arrive during the dropout, the TFRC sending is

halved once. This cannot be seen in the graph because the halving coincides with the packet burst. As soon as receiver reports arrive again, the sending rate is readjusted to the expected bitrate. The sudden increase in RTT and the experienced loss events cause a decrease in the expected bitrate, but this decrease is much smaller than the corresponding decrease in the TCP rate.



a) Connection dropouts

Figure 4.25: Unusual conditions

TFRC recovers faster than TCP because TCP delays packet retransmission after a timeout event. However, when TCP performs slowstart, it can claim available bandwidth faster than TFRC and the TCP bitrate exceeds the TFRC rate soon after slowstart. The overall TFRC behavior under such circumstances is good as long as the dropout is a transient condition. As mentioned before, the goal of the TFRC protocol is not to follow TCP's short-term throughput. When the dropout is caused by persistent congestion, loss events will occur eventually. This will cause a decrease in the expected bitrate, until that rate matches the network conditions again. Because of the smoothing inherent in the TFRC protocol, the decrease happens over a longer time interval.

4.10 Prediction quality of the loss estimate

The size of the loss history determines how fast TFRC adapts to changes in the loss rate. A large loss history results in very stable protocol behavior and a good resilience against noise, while the responsiveness to changed network conditions is low. A small loss history size on the other hand provides a high responsiveness to changes in the loss rate while the loss estimate is more susceptible to noise and less stable. Thus, it is necessary to find a loss history size that provides an acceptable tradeoff between responsiveness and stability.

One measure of the effectiveness of the loss estimate is its ability to predict the immediate future loss rate. The loss prediction error is defined as the difference of loss estimate and future loss rate. In figure 4.26, the average loss prediction error and the average standard deviation of the loss prediction error of all the Internet experiments is shown. The left part of the graph contains the prediction error for a loss history where all loss intervals receive equal weights, while the right part of the graph shows the same values for a loss history

where old intervals receive less weight.[12]

A small loss history size causes a high prediction error when the immediate past loss rate does not match the immediate future loss rate. However, when the loss history size gets too large it does no longer adequately represent short-term changes in the loss rate, which also results in a high prediction error. A number of eight to sixteen loss intervals provides the best prediction of the future loss rate for decreasing interval weights as well as constant interval weights. The absolute error is slightly smaller when decreasing weights are used. Although the loss prediction accuracy is not the only criterion for the quality of the loss estimation method, the results presented in this section provide experimental confirmation that using eight loss intervals with decreasing weights is a reasonable choice for the composition of the loss history.

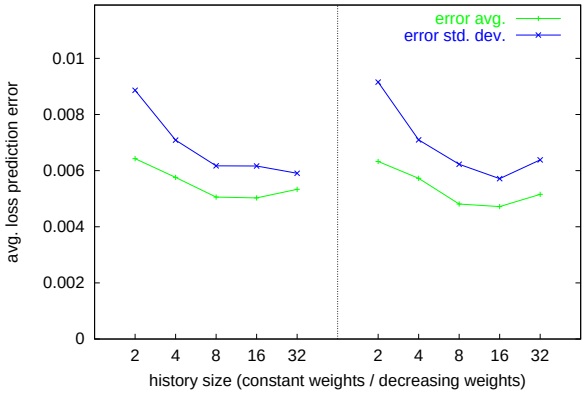


Figure 4.26: Quality of the loss estimation with different loss history sizes

4.11 Comparison of experiment data to the model

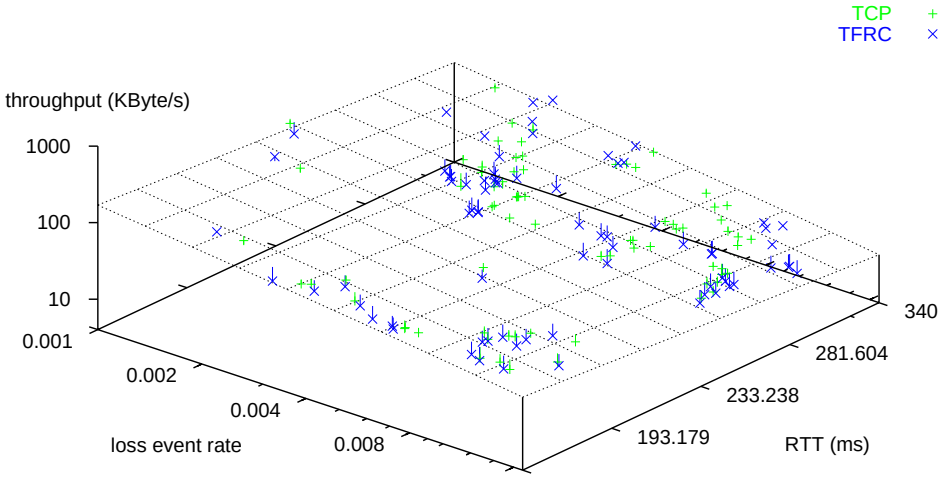
To provide a sanity check for the functioning of the protocols, figures 4.27 and 4.28 map the average throughput of TFRC and TCP flows in the network experiments on the graph of the throughput given by the TCP model under the same average network conditions. Measurement points represent the average conditions of one test run. They are connected to the plane of the model throughput with a line to indicate over- or underutilization of bandwidth.

In the UCL and UMANN experiments, TCP and TFRC throughput corresponds very well to the throughput predicted by the TCP model. TFRC throughput is slightly lower than the model throughput (which corresponds to a minimal underutilization of bandwidth by TFRC during these experiments). Model throughput and TFRC throughput can differ, since model throughput is calculated for the average network conditions of the experiment, while TFRC has to estimate the current network parameters from the previous values of these parameters. The estimate depends on the measurement method and the amount of smoothing that is used. For the UMASS experiments, TFRC throughput again corresponds well to the model throughput. TCP only matches model throughput when the loss rate is low. For higher loss rates, the effect of the too aggressive TCP timeout value of the Solaris TCP implementations becomes visible. TCP throughput is constantly less than the throughput predicted by the model. The graph also shows, that while the average RTT

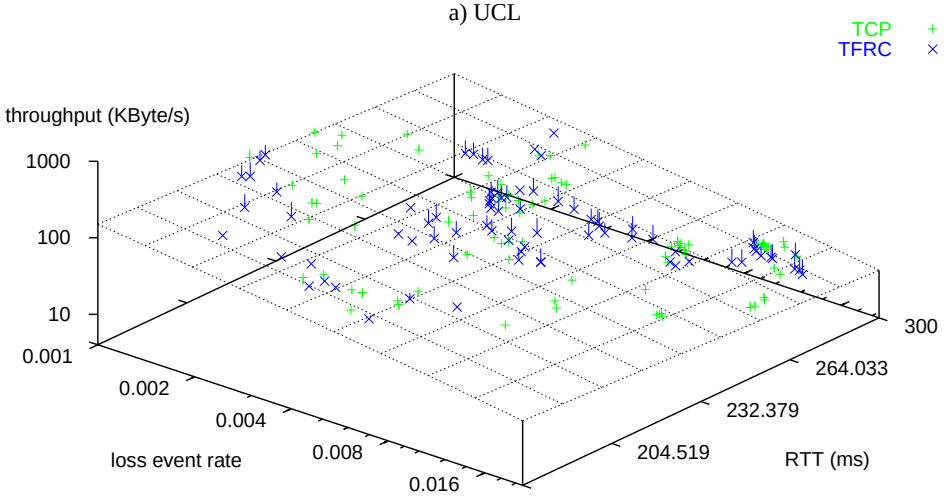
[12]The distribution of the weights for the loss intervals is explained in detail in section 3.4.1.

is lower in the UMASS experiments, the average loss rate is a multiple of the loss rates experienced in the UCL and UMANN experiments.

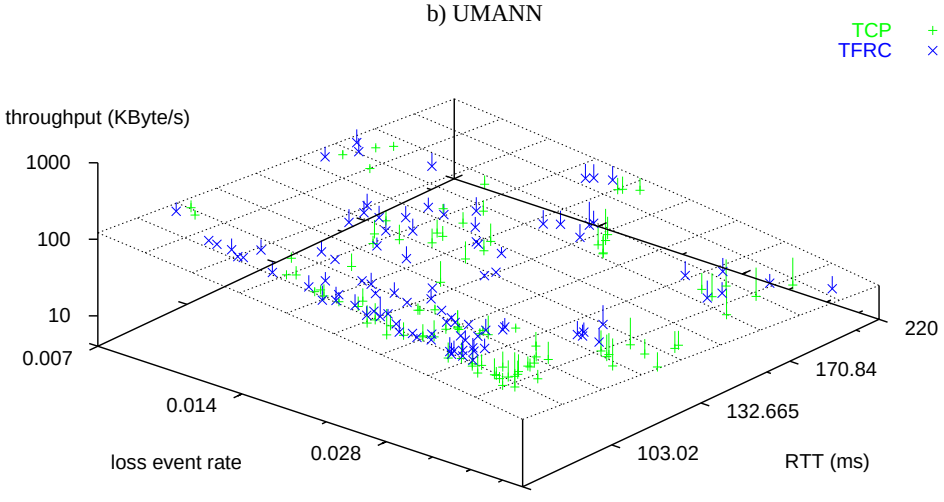
In the Nokia experiments, TCP performance is poor when the loss rate is very high. This is a result of TCP's difficulty to cope with scarce buffer space when competing against TFRC without ISM, as discussed in section 4.4.3. The cable modem experiments show a good correspondence of experiment conditions and the throughput estimates of the TCP model. Due to the large buffer of the cable modem connection, the throughput distribution is biased in favor of TCP. The buffer delay is very high and causes RTT values in the range of several seconds while the loss rate is comparatively low. TCP achieves better performance than TFRC under such conditions.



a) UCL

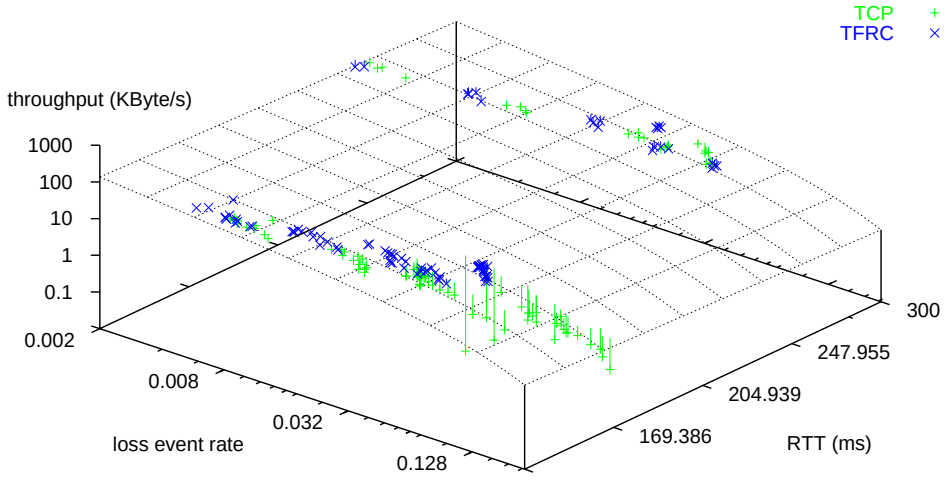


b) UMANN

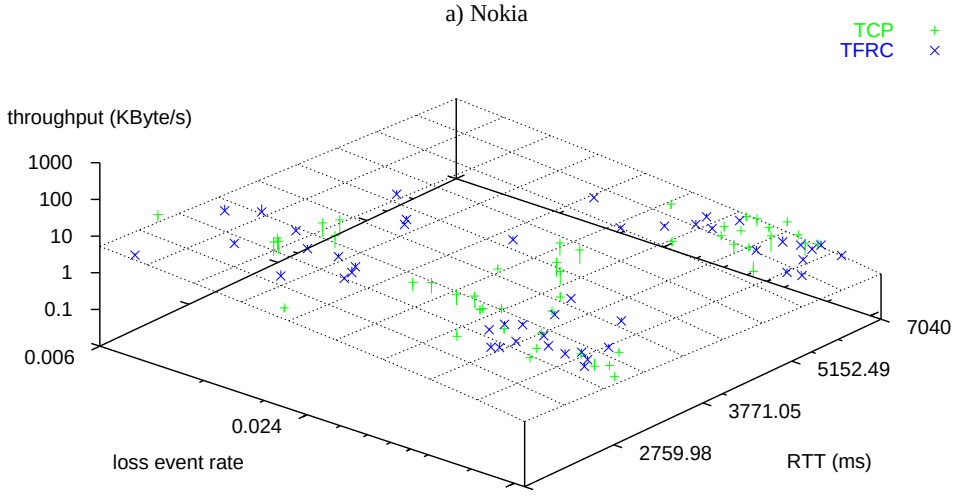


c) UMASS

Figure 4.27: Comparison of data and model
(high level of statistical multiplexing)



a) Nokia



b) Modem

Figure 4.28: Comparison of data and model
(low level of statistical multiplexing)

Chapter 5

Related Work

Congestion control is a research area that has been extensively explored in the past. In this chapter, we discuss related work on unicast congestion control protocols and compare characteristics of these protocols to the TFRC protocol. Work on multicast congestion control is addressed only as far as it is related to issues concerning unicast congestion control.

5.1 Related protocols

The protocols discussed here can be subdivided into three classes according to the type of congestion control being used. *Window-based* congestion control protocols employ a congestion window in the same manner as TCP and in general show similar behavior. *TCP-like* protocols adjust their sending rate according to TCP's Additive Increase Multiplicative Decrease strategy without using a congestion window. The TFRC protocol presented in this thesis belongs to the third category of congestion control mechanisms, the *equation-based* congestion control protocols. As described in previous chapters, equation-based congestion control mechanisms use an equation to determine the appropriate sending rate. Usually, the equation is derived from a model of competing traffic, for example TCP.

5.1.1 Window-based protocols

- Golestani presented a general framework for both window-based and rate-based flow control algorithms in [Gol98]. Based on these foundations, a unicast congestion control scheme for IP-networks was developed, called the *minimum cost flow control* algorithm (MCFC). The *exact* realization of the algorithm needs additional router support, while a *coarse* realization is designed to work in today's Internet. The MCFC has some improvements over TCP but shows similar overall behavior. In [GS99], the above framework is extended to support multicast traffic with modifications for distinct windows at the receivers and for hierarchical feedback aggregation.
- In [JE96], the authors directly use TCP's congestion control algorithm to modify the packet rate of a video stream. The sender maintains a congestion window and translates its size to an output rate for the video codec.

- The approach used for *TCP Emulation at the Receivers* (TEAR) in [OR99] is similar to the previous method in that a congestion window is maintained and the size of the window is used to set the sending rate (although with a more general focus). To be able to employ this mechanism with multicast as well as unicast, the congestion window is located at the receiver instead of the sender. Timeouts and triple duplicate ACKs are emulated by the receiver and used to modify the size of the congestion window $cwnd$ in the same manner as TCP. The receiver estimates the RTT and computes the sending rate as $cwnd/RTT$. Previous work by the authors on the MTCP protocol [RBR98] - a multicast version of TCP - is not dicussed here because of its solely multicast nature.

5.1.2 TCP-like protocols

- The *Loss-Delay Based Adjustment Algorithm* (LDA) presented in [SS98] mimics TCP's Additive Increase Multiplicative Decrease mechanism. The protocol must be used in conjuction with the Real-Time Transfer Protocol [SCFJ96] since its rate adjustment decisions are based on feedback information provided by the Real-Time Control Protocol (RTCP). In addition, a simple packet-pair mechanism is used to estimate the bottleneck bandwidth. In periods of no packet losses the sending rate is increased using an Additive Increase Rate (AIR) that depends on the ratio of current sending rate to bottleneck bandwidth. When RTCP indicates losses, the sending rate is exponentially decreased in proportion to the number of lost packets.
- In the *Rate Adaption Protocol* (RAP) presented in [RHE99] each data packet is acknowledged by the receiver. The ACKs are used to detect packet loss and infer the RTT. Redundant information in the acknowledgements provides resilience against the loss of an ACK. When the protocol experiences congestion in form of the loss of a data packet, it halves the sending rate. In periods without congestion the sending rate is increased by one packet per RTT. These decisions on rate increase or decrease are made once per RTT. To provide additional fine-grained delay-based congestion avoidance, the ratio of a short-term RTT average and a long-term RTT average is used to modify the inter-packet gap between consecutive data packets.

5.1.3 Equation- or model-based protocols

- The concept of congestion control using an equation which is based on a model for TCP throughput was first proposed in [MF97]. Although the paper does not give specific details about a congestion control protocol, it outlines important characteristics that such a protocol should exhibit.
- The congestion control mechanism described in [PKT99] is similar to the TFRC protocol insofar as it uses the complex model for TCP [PFTK98] to adjust its sending rate in a TCP-friendly manner. However, the way in which the necessary parameters are gathered differs considerably from TFRC. Time is divided into *rounds* with a fixed duration M. The model parameters are recomputed for each round. When no packets are lost during a round, the sending rate is doubled in the next round. Otherwise, the sending rate is set to the rate given by the TCP model. The protocol relies strongly on the duration M of the recomputation interval. In this protocol, each data packet must be acknowledged by the receiver.
- A similar scheme for multicast which is based on the complex TCP model was presented in [WC98]. The interval for updating the control parameters is a fixed number of packets. Model parameters are smoothed by using an exponentially weighted moving average. The sending rate is set to the

throughput given by the equation without further increase or decrease constraints.

- RLC, a receiver driven layered multicast congestion control algorithm, is presented in [VRC98]. The data is split into layers and the cumulative transmission rate to a receiver is proportional to the number of layers the receiver subscribes. By only permitting join and leave decisions of the receivers at so-called synchronization points and by using an exponential distribution of the bandwidth of the layers, the protocol throughput is related to $1/\sqrt{l}$ where l is the packet loss rate. Thus, the dependency of the protocol throughput on the loss rate is similar to the dependency of TCP throughput to the loss rate in the simple equation. No explicit feedback is necessary from the receivers to the sender.
- In [TZ99], the authors present a congestion control scheme coupled with a rate-adaptive video codec to provide TCP-friendly streaming video. The simple equation is used for rate shaping and again the interval for updating the model parameters is fixed. No smoothing is used for the parameters.

5.2 Comparison

Pure AIMD schemes as well as rate-based schemes based on the simple TCP model [MF97] do not take TCP timeouts into account. Consequently, they are unfair to TCP in environments with a high loss rate where TCP loses its self-clocking. The same holds for RLC where the throughput is inversely proportional to the square root of the loss rate. Furthermore, the RLC throughput is not explicitly dependent on the RTT. Protocol fairness is biased against TCP on connections with a high propagation delay. A strong argument in favor of RLC is that RLC does not rely on feedback messages from the receivers to the sender. The protocol can be employed in multicast environments where no back channel for the communication end-points is present.

Window based schemes adjust their rate in a manner very similar to TCP (or the specific TCP flavor they emulate). While this is TCP-friendly, the protocol also exhibits some of the negative aspects of TCP traffic. As discussed in the introduction, the sudden rate changes by a window mechanism make this approach unsuitable for some applications. In addition, the coupling of error recovery and rate control often used in conjunction with a window mechanism does not always meet application requirements.

During the development process of the TFRC protocol it became evident that static measurement intervals for rate-based schemes as in [PKT99, WC98, TZ99] were not well suited to capture the dynamics of network behavior over a wide range of conditions and could result in unfairness to competing flows.

Chapter 6

Conclusions and Outlook

In this thesis, we have developed and evaluated a rate-based unicast congestion control protocol. The protocol is intended to provide an alternative to TCP for applications that are inherently rate-based, have different error recovery requirements, and that need a more stable sending rate.

6.1 Synopsis of the TFRC protocol

TFRC uses packet loss as the main indicator of congestion in the network. This is supported by the results of the protocol experiments because the bandwidth share of the flows was mostly determined by the loss rate. An accurate loss estimate is crucial for fair protocol behavior. For this reason, we employ a sophisticated loss estimation method that offers good responsiveness to permanent changes in the congestion level. At the same time, the method provides a stable sending rate and exhibits resilience against noise in the congestion signal.

The complex TCP model employed by the TFRC protocol uses a long-term notion of the RTT. For this reason, we use an EWMA filter to obtain a smooth and stable RTT estimate from the RTT samples. The model does not reflect the dependency of the RTT and the sending rate in environments with a low level of statistical multiplexing.

Nevertheless, a form of delay-based congestion control can improve protocol behavior in environments with a low level of statistical multiplexing. As mentioned before, the TCP equation does not accurately model these conditions. To take advantage of RTT changes that indicate an increase or decrease in the congestion level, the TFRC protocol uses Inter-packet Space Modulation (ISM). ISM, as presented in section 3.6.4, modifies the gap between consecutive data packets (i.e. the short-term sending rate). Modifications occur on the same timescale as the variations in the buffer utilization; they do not affect the long-term sending rate of the protocol. In Internet experiments with a high level of statistical multiplexing, the RTT was found to be fairly steady and independent of the current sending rate. In conclusion, ISM improves protocol stability when the level of statistical multiplexing is low, but does not noticeably change protocol behavior when the level of statistical multiplexing is high. In such an environment, the TCP equation does an adequate job.

6.2 Possible protocol adjustments

There is no performance penalty for TFRC flows as far as throughput is concerned. However, the computation of the TCP equation involves floating point arithmetic and square root calculations. For some receivers, these calculations may consume too much processing power. A lookup table listing throughputs (as calculated by the equation) for various loss rate - RTT combinations can be used instead.

By the use of the terms 'sender' and 'receiver', we imply that communication is one way; however, this is not necessarily the case. Only small modifications to TFRC and the packet format[1] are necessary to enable duplex data transfer. The amount of control traffic can be reduced by piggybacking receiver reports (i.e. combining a data packet with a report).

The parameters of the TFRC protocol are adjusted to assure fair sharing of resources with other flows in the Internet. In networks where flows of different protocols do not share the same resources, that is flows compete only against flows of the same protocol type, other control parameter settings may improve protocol performance. The same applies for networks that support Quality of Service (QoS) (e.g. via MPLS [AMA+99] or Diffserv [BBC+98]). Further research is needed to assess protocol behavior and necessary parameter adjustments in such environments. Since the protocol module with the TCP equation can be easily exchanged, an equation based on a model other than TCP can be used.

6.3 Issues not addressed in the thesis

Sending below the fair share of bandwidth

Currently, the TFRC protocol assumes that applications will fully utilize their fair share of bandwidth. Since in reality this is not always the case, the TFRC protocol will transmit data at less than the maximum bitrate. A lower sending rate is likely to cause a lower loss event rate and RTT. Based on the reduced parameters, the model overestimates the fair share of bandwidth. As long as the application does not attempt to increase the sending rate, overestimating the available bandwidth is not a problem. However, when the application increases the sending rate, the increase must be limited to the "true" fair share of bandwidth, not the overestimated value. Overestimation can be prevented by decaying the bandwidth estimate in periods of underutilization. This ensures that the sending rate does not exceed the fair share of bandwidth. Since the current sending rate is lower than the "true" fair share, it is only necessary to decay the estimate to the actual sending rate. The decayed expected bitrate will adapt smoothly to the fair share of bandwidth once the protocol starts to send at full speed again.

Performance improvements by Random Early Detection (RED) and Early Congestion Notification (ECN)

One problem inherent in rate-based congestion control schemes is that they rely on packet loss to limit the sending rate. TFRC increases its rate until packet loss occurs. In an environment with a high level of statistical multiplexing, there is always at least a small amount of congestion present in the network that causes the necessary packet loss to limit the rate of TFRC. However, in an environment with a low level of

[1] The complete packet header for TFRC is presented in appendix A.

statistical multiplexing packets are only dropped when there are no more resources available in the network. Thus, TFRC constantly causes a high buffer utilization that is unnecessary. The same amount of throughput could be achieved with a lower buffer utilization and fewer dropped packets. When TCP is not the only flow on a link, it exhibits similar behavior to TFRC but the maximum amount of buffer space that is used is limited. A good solution to the problem is to use RED queues [FJ93] instead of drop tail queues in the routers. In RED queues, packets can be dropped before the buffer is full. Thus, the router avoids having to discard a large amount of packets when the buffer overflows. Buffer utilization is reduced and the overall performance of the network improves.

TFRC can easily be adapted to support new technologies like Early Congestion Notification (ECN) [RF99] where congestion is not signaled by dropping packets but by marking them. Packet drops are an exception rather than the rule and only occur when flows do not adequately respond to the ECN signal.

Synchronization of flows

In an environment where the loss rate is very low, the loss history comprises a large number of packets. If congestion occurs only at very few routers, TFRC flows that share the same path over these routers have a similar loss history. This can cause TFRC flows to synchronize. The flows increase and decrease their bandwidth simultaneously. Synchronization is harmful to the network when the simultaneous rate increase of many flows causes congestion. However, synchronization only appears in TFRC when the loss rate is very low. Timely feedback from the receivers dampens the increase before congestion at a router can become critical. When the loss rate increases and synchronization may be critical to the network, already small differences in the experienced loss patterns will cause the flows to desynchronize. Synchronization of TFRC flows was only present in a very small number of experiments and in none of the experiments did the synchronization cause additional congestion.

Protection against misbehaving receivers

The TFRC protocol relies on cooperative receivers. It has no built-in mechanisms to prevent misbehaving receivers from sending false reports. Such reports can be used to cause the sender to increase its rate above the fair share of bandwidth which results in a disproportionally high throughput for one flow at the expense of competing flows. The following measures should be taken to prevent this form of attack:

- Move computation of the TCP equation to the sender. In addition, the necessary model parameters also have to be gathered there.
- Modify timestamp value in packet headers to prevent the receiver from changing the value of the sender's RTT estimate. Either the timestamp has to be encrypted so that simple addition of a number does not lead to a valid timestamp, or the RTT measurement method has to be based on packet identifiers rather than timestamps. The sender can assign a unique random identifier to each packet (sequence numbers cannot be used). The receiver has return these identifiers in the receiver reports. By storing the sending time of the packet together with the identifier, the sender can then compute an RTT sample as soon as a report with the corresponding identifier arrives back at the sender.
- Furthermore, the sender has to make sure that the loss estimate is correct. This is more difficult since packet loss can only be detected by the receiver. By letting the receiver prove to the sender what packets it received, the sender can prevent concealment of packet loss by the receiver. The receiver

computes a validation number as the sum or binary *exclusive or* over the identifiers of the packets it received. The validation number together with a list of the sequence numbers of the received packets is sent back to the sender. The sender keeps a table with sequence numbers and identifiers. By looking up the identifiers that correspond to the list of received packets, the sender can then recompute the validation number. Only if the validation number matches the number reported by the receiver, does the sender uses the list of packets to update its loss history and the loss estimate. Otherwise, the sender assumes that the receiver attempted to fake the list of received packets and accordingly reduces the sending rate. Similar modifications to the TCP protocol have been proposed in [SCWA99].

The problem of a misbehaving sender cannot be solved by the modification of the protocol but only by additional support from the routers.

6.4 Multicast congestion control

Multicast IP allows senders to transmit data efficiently to several receivers at the same time. Multicast network paths have a tree structure with the sender as the root of the tree. Instead of having to send multiple copies of the same data, the data is sent only once and is duplicated in the network when necessary.[2]

Although multicast congestion control is much more complex than its unicast counterpart, many of the mechanisms developed for TFRC can be used in a multicast version of the protocol. In particular, it is not necessary to modify the loss measurement method. Also the sender's rate increase and decrease policies are well suited for multicast.

To ensure that a multicast congestion control protocol cannot harm the network, the sending rate has to be adjusted to the lowest expected bitrate of all the receivers. This guarantees that the rate of the multicast session does not exceed its fair share of bandwidth anywhere in the network. However, with this approach a slow receiver can impair the performance of the whole multicast session. Mechanisms are necessary to prevent that a joining receiver overly reduces the overall sending rate. This limits the "allowed" amount of heterogeneity of a session. Since the definition of an acceptable sending rate is application specific, the join and leave decisions should be made on the application layer.

6.4.1 Scalability

Scalability is a major concern for multicast protocols. To enable the sender to handle large numbers of receivers, all protocol functionality that can be shifted to the receivers should be shifted to the receicers. All computationally intense calculations in the TFRC protocol are already made at the receiver, which facilitates adaption of the protocol to multicast.

However, decentralization of time consuming protocol functions is not the only criterion for scalability. When a large number of receivers send reports back to the sender, there can be a feedback implosion. The sender can only process a limited number of feedback messages before it is no longer able to perform other necessary tasks. Feedback implosion can be prevented in two ways, either by aggregating feedback at the intermediate nodes between sender and receiver or by allowing only selected receivers to send a report.

[2] For a detailed overview of IP multicast see [Dee91].

Feedback aggregation must be supported by the nodes in the multicast tree while feedback suppression can be used without a modification of the existing multicast IP protocol. An elegant way to provide feedback suppression is to have the receiver wait a specific time before multicasting a report to all other receivers as well as the sender. A report is *only* sent when no other report indicating a lower expected bitrate was received during the waiting period. By using exponential timers with a timeout value that is dependent on the expected sending rate, the probability can be increased that only a small number of reports are sent. In most cases, a receiver receives a report indicating a lower expected bitrate before this receiver sends its own report, provided another receiver has a lower expected bitrate. In the ideal case, only the report with the lowest bitrate reaches the sender and the sender can then use the expected bitrate of this report to adjust the sending rate. There is a tradeoff between responsiveness and the number of reports that the sender is likely to receive. The longer the waiting period, the higher the probability that only very few reports reach the sender. This comes at the expense of a longer delay in the feedback loop from the receiver to the sender. The increased delay in the control system changes protocol dynamics. In general, a multicast version of the protocol would have to be slightly more conservative than its unicast counterpart to ensure safe operation.

6.4.2 Further issues with multicast congestion control

The slowstart mechanism of TFRC requires a timely response by the receiver as soon as the first packet loss occurs. As mentioned before, timely response cannot always be guaranteed in a multicast environment. Alternatives to the current slowstart mechanism need to be explored.

RTT estimation by the sender works only for small sets of receivers. A different approach is necessary to ensure scalability of the protocol to thousands of receivers. Several proposals for efficient RTT estimation in multicast environments have been proposed in the literature. For example, in [BG99] the authors present a scheme to estimate RTTs when receivers are organized hierarchically in a multicast tree. The RTTs between child nodes and parent nodes in the tree hierarchy are measured. The RTT to a specific receiver consists of the sum of the RTTs of all intermediate nodes along the path in the multicast tree from the sender to the receiver. Future research should address the suitability of different RTT estimation schemes for the multicast congestion control protocol.

6.5 Conclusions

TFRC meets the protocol requirements formulated in the introduction. The extensive experiments in chapter 4 indicate that TFRC exhibits good inter- and intra-protocol fairness. TFRC achieves the same long-term throughput over a wide range of network conditions as a conformant TCP flow. On shorter timescales, TFRC throughput varies considerably less than TCP throughput. The higher stability makes the protocol suitable for traffic where sudden rate changes are undesirable. The TFRC protocol shows promosing potential as an alternative to TCP. It exhibits superior behavior to comparable rate-based congestion control protocols that have been discussed in the literature so far. Furthermore, rate-based congestion control is a promising basis for large scale multicast transport protocols.

Bibliography

[MF97] J. Mahdavi and S. Floyd, “TCP-Friendly Unicast Rate-Based Flow Control”, Note sent to the end2end-interest mailing list, January 1997

[OKM96] T .Ott, J. Kemperman, and M. Mathis, “The Stationary Behavior of Ideal TCP Congestion Avoidance”, August 1996

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T .Ott, “The Macroscopic Behavior of the Congestion Avoidance Algorithm”, Computer Communications Review, volume 27, number 3, July 1997

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP Throughput: A Simple Model and its Empirical Validation”, Proceedings of ACM Sigcomm, Vancouver, Canada, 1998

[PFT99] J. Padhye, V. Firoiu, and D. Towsley, “A Stochastic Model of TCP Reno Congestion Avoidance and Control”, Technical Report 99-02, Department of Computer Science, University of Massachusetts, Amherst, February 1999

[RR99] S. Ramesh and I. Rhee, “Issues in TCP Model-Based Flow Control”, NCSU Technical Report TR-99-15, November 1999

[HFW99] M. Handley, S. Floyd, and B. Whetten, “Strawman Specification for TCP-Friendly (Reliable) Multicast Congestion Control (TFMCC)”, Unpublished Manuscript, June 1999

[HF] M. Handley and S. Floyd, “TCP-Friendly Simulations”, Unpublished Manuscript

[PKT99] J. Padhye, J. Kurose, D. Towsley, and R .Koodli, “A Model Based TCP-Friendly Rate Control Protocol”, Proceedings of NOSSDAV’99, 1999

[WC98] B. Whetten and J. Conlan, “A Rate Based Congestion Control Scheme for Reliable Multicast”, Technical White Paper, November 1998

[VRC98] L. Vicisano, L. Rizzo, and J. Crowcroft, “TCP-like Congestion Control for Layered Multicast Data Transfer”, IEEE Infocom 98, March 1998

[MJV96] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven Layered Multicast”, Proceedings of SIGCOMM ’96, Stanford, CA, August 1996

[RHE99] R. Rejaie, M. Handley, and D. Estrin, “RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet”, Proceedings of INFOCOMM ’99, 1999

[SS98] D. Sisalem and H. Schulzrinne, “The Loss-Delay Adjustment Algorithm: A TCP-Friendly Adaptation Scheme”, Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), (Cambridge, England), July 1998

[CHKW98] D.M. Chiu, S. Hurst, M .Kadansky, and J. Wesley, “TRAM: A Tree-based Reliable Multicast Protocol”, Sun Microsystems Laborities Technical Report Series, TR-98-66, July 1998

[TZ99] W. Tan and A. Zakhor, “Real-time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol”, IEEE Trans. Multimedia, vol. 1, no. 2, pp. 172-186, June 1999

[RBR98] I. Rhee, N. Ballaguru, and G. Rouskas, “MTCP: Scalable TCP-like Congestion Control for Reliable Multicast”, TR-98-01, Department of Computer Science, NCSU, January 1998

[OR99] V. Ozdemir and I. Rhee, “TCP emulation at the receivers (TEAR)”, Presentation at the RM meeting, November 1999

[JE96] S. Jacobs and A. Eleftheriadis, “Providing Video Services over Networks without Quality of Service Guarantees”, Proceedings of the WWW Consortium Workshop on Real-Time Multimedia and the Web, Sophia Antipolis, France, October 1996

[FJL97] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, “A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing”, IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 784-803, December 1997

[Gol98] J. Golestani, “A Class of End-to-End Congestion Control Algorithms for the Internet”, Proceedings of the International Conference on Network Protocols, 1998

[GS99] J. Golestani and K. Sabnani, “Fundamental Observations on Multicast Congestion Control in the Internet”, Proceedings of the Conference on Computer Communications (IEEE Infocom), March 1999

[BG99] A. Basu and J. Golestani, “Architectural Issues for Multicast Congestion Control”, Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), June 1999

[Mon97] T. Montgomery, “A Loss Tolerant Rate Controller for Reliable Multicast”, NASA IV&V Technical Report NASA-IVV-97-011, August 1997

[Pos80] J. Postel, “User Datagram Protocol”, RFC 768, Network Information Center, SRI International, Menlo Park, CA, August 1980

[Pos81a] J. Postel, “Internet Protocol”, RFC 791, Network Information Center, SRI International, Menlo Park, CA, September 1981

[Pos81b] J. Postel, “Transmission Control Protocol”, RFC 793, Network Information Center, SRI International, Menlo Park, CA, September 1981

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, January 1996

[FF96] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, Computer

Communication Review, vol. 26, no. 3, pp. 5-21, July 1996

[Dee91] S. Deering, “Multicast Routing in a Datagram Internetwork”, PhD thesis, Electrical Engineering Dept., Stanford University, December 1991

[Pad00] J. Padhye, “Towards a Comprehensive Congestion Control Framework for Continuous Media Flows in Best Effort Networks”, PhD thesis to appear, University of Massachusetts, 2000

[Jac88] V. Jacobsen, “Congestion Avoidance and Control”, Proceedings of ACM Sigcomm, August 1988

[Jac90] V. Jacobsen, “Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno”, Proceedings of the Eighteenth Internet Engineering Task Force, University of British Columbia, Vancouver, B.C., 1990

[Kes91] S. Keshav, “Congestion Control in Computer Networks”, PhD Dissertation, Department of EECS at UC Berkeley, August 1991

[FF99] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet”, IEEE/ACM Transactions on Networking, August 1999

[CJ89] D. Chiu and R. Jain, “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks,” Journal of Computer Networks and ISDN, vol. 17, no. 1, pp. 1-14, June 1989

[Jai92] R. Jain, “Myths about Congestion Management in High Speed Networks”, Internetworking: Research and Experience, vol. 3, pp. 101-113, 1992

[JCH84] R. Jain, D.M. Chiu, and W. Hawe, “A Quantitative Measure of Fairness ans Discrimination for Resource Allocation in Shared Computer System”, Digital Equipment Corp. Eastern Research Lab, DEC-TR-301, September 1984

[BCC+98] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, “Recommendations on Queue Management and Congestion Avoidance in the Internet”, RFC 2309, April 1998

[Ste98] R. Stevens, “Unix Network Programming, Volume 1”, 2nd ed., Prentice Hall, 1998

[Ste94] R. Stevens, “TCP/IP Illustrated, Volume 1. The Protocols”, Addison-Wesley Publishing Company, 1994

[WS95] G. Wright and R. Stevens, “TCP/IP Illustrated, Volume 2. The Implementation”, Addison-Wesley Publishing Company, 1995

[AP99] M. Allman and V. Paxson, “On Estimating End-to-End Network Path Properties”, ACM SIGCOMM ’99, Cambridge, MA, September 1999

[PF97] V. Paxson and S. Floyd, “Why We Don’t Know How To Simulate The Internet”, Proceedings of the 1997 Winter Simulation Conference, December 1997

[MF99] S. McCanne and S. Floyd, “UCB/LBNL Network Simulator - ns (version 2)”, *URL:* http://www-mash.cs.berkeley.edu/ns, 1999

[Riz97] L. Rizzo, “Dummynet: A simple approach to the evaluation of network protocols”, ACM Computer Communication Review, Jan. 1997

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment Options”, RFC 2018, October 1996

[RF99] K. Ramakrishnan and S. Floyd, “A Proposal to add Explicit Congestion Notification (ECN) to IP”, RFC 2481, January 1999

[FJ93] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, vol. 1, no. 4, pp. 397–413, August 1993

[BBC+98] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An Architecture for Differentiated Services”, RFC 2475, December 1998

[AMA+99] D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell, J. McManus, “Requirements for Traffic Engineering Over MPLS”, RFC 2702, September 1999

[CMT98] K. Claffy, G. Miller, and K. Thompson “The nature of the beast: recent traffic measurements from an Internet backbone”, in Proceedings of INET’98, ISOC, Washington, DC, 1998

[BG87] D. Bertsekas ans R. Gallager, “Data Networks”, Prentice-Hall, 1987

[MR99] L. Massoulie and J. Roberts, “Bandwidth Sharing: Objectives and Algorithms”, Proceedings of INFOCOM 99, 1999

[Kel97] F. Kelly, “Charging and rate control for elastic traffic”, European Transactions on Telecommunications, vol. 8, pp. 33-37, 1997

[BG99] A. Basu and J. Golestani, “Estimation of Receiver Round Trip Times in Multicast Communications”, 1999

[SCWA99] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, “TCP Congestion Control with a Misbehaving Receiver”, to appear in Computer Communications Review, vol. 29, no. 5, October 1999

[Pax97a] V. Paxson, “Measurements and Analysis of End-to-End Internet Dynamics”, Ph.D. dissertation, Computer Science Division, University of California, Berkeley, April 1997

[Pax97b] V. Paxson, “End-to-End Routing Behavior in the Internet”, IEEE/ACM Transactions on Networking, Vol.5, No.5, pp. 601-615, October 1997

[JLM89] V. Jacobsen, C. Leres, and S. McCanne, “tcpdump”, *URL:* ftp://ftp.ee.lbl.gov, June 1989

[MD90] J. Mogul and S. Deering, “Path MTU Discovery”, RFC 1191, November 1990

Appendix A

Packet Format

Figure A.1 shows the packet format used in the TFRC implementation. The *sequence number* is unique for each the packet. It is increased by one for consecutive packets to allow the receiver to detect missing packets. The unit for the *timestamp* is milliseconds. It is obtained by adequately scaling the system timer. The *round-trip time* is computed from the timestamp values of the receiver reports and also measured in milliseconds. The *sending rate* in KByte/s is only sent for monitoring reasons and can be left out in an actual implementation of the protocol. The value of the *mode* field corresponds to the different protocol states:

- Init RTT: 0
- Slowstart: 1
- Normal Congestion Control: 2
- Forced Congestion Control: 4

In the receiver report, the values for the sequence number and the timestamp are taken from the last data packet that arrived. This enables the sender to calculate the RTT from the timestamp value. The *received rate* gives feedback on the actual rate with which the receiver was getting the data packets since the last receiver report and the *expected rate* is the rate determined by the TCP model for the current network conditions. Both are measured in KByte/s.

No attention was paid to keeping the header size small. Each header field has a size of 32 bits. For an actual deployment of the protocol, the header size could be reduced. For example, 16 bits are an adequate size for the RTT sample and with appropriate scaling, even 8 bits may be sufficient. Since only four modes are presently used, the mode field can be reduced to 2 bits. While a reduction of the size of the bitrate field would be possible, it is not recommended to allow for fine grained rate control over a wide range of network conditions. The headers also do not include placeholder fields for future extensions of the protocol which are important for an end product protocol version.

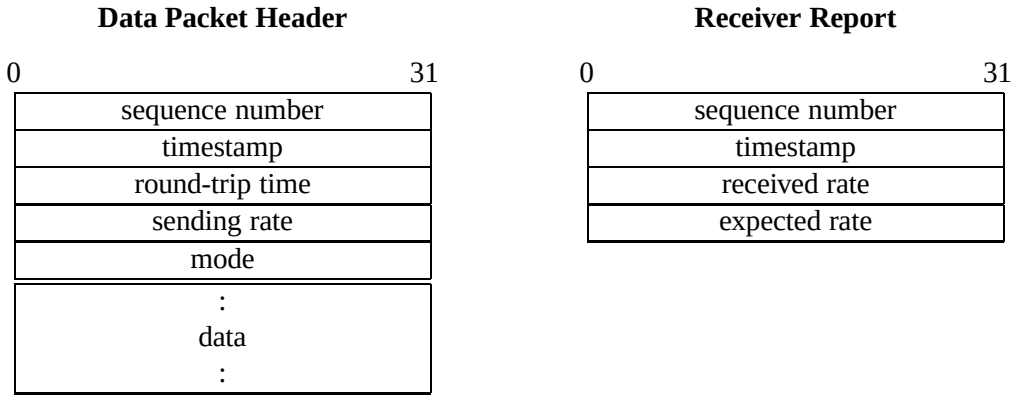


Figure A.1: Header structure

Appendix B

Overview of the protocol states

The protocol operates in the following modes:

Init RTT mode: To determine the initial RTT, packets are sent at a very low bitrate until the first receiver report arrives and the RTT can be initialized. The mode is switched to *slowstart*.

Slowstart mode: The sending rate is doubled every RTT to determine the initial sending rate for the congestion control mode. The sender terminates slowstart when a receiver report is missing or a loss event occurs. When the sender experiences a send error it changes the mode to *force congestion control*.

Force congestion control mode: In this mode, the sender is prevented from reentering *slowstart mode* even when receiver reports arrive that indicate no packet loss. Only when loss is reported the mode is changed to normal congestion control.

Congestion control mode: The sending rate is determined by the sender based on the RTT and loss estimate using the accurate TCP model. When a receiver report indicates that no loss event happened yet, *slowstart mode* is reentered.

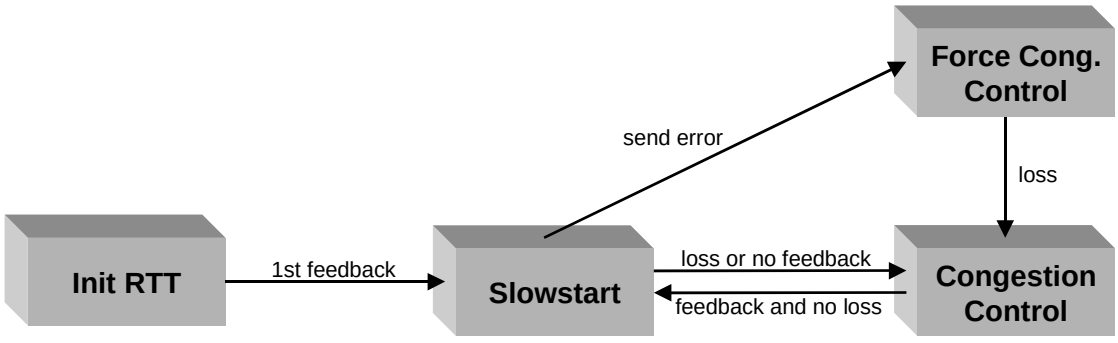


Figure B.1: Protocol state diagram

Appendix C

Symbols

$t_{RTTsample}$	the time it takes a packet to travel from sender to receiver and back to the sender
t_{RTT}	EWMA of the RTT samples
t_{RTT}^{sqrt}	EWMA of the square root of the RTT samples
t_{RTTvar}	EWMA of the variance of the RTT samples
t_{RTO}	time until an unacknowledged packet is considered lost (timeout value)
t_{now}	current time
t_{ts}	timestamp value in the TFRC packet header
Δ_p	time interval between two consecutive data packets
Δ_p^{ISM}	time interval between two consecutive data packets with inter-packet space modulation
s	size of a data packet (including the header)
l	probability that a packet is lost on its way from source to destination
ppl	number of packets between consecutive loss events (loss interval size)
l_{act}	probability that a packet is lost given that it is the first packet in a round or all of the previous packets of that round are not lost
l_{est}	number of loss events per packets sent (estimate for l_{act})
l_{exp}	loss event rate given by the TCP equation for the current sending rate and RTT
h	size of the loss history for the loss rate estimation
b_{act}	actual sending rate
b_{exp}	sending rate given by the TCP equation for the current RTT and loss estimate
b_{rep}	current receive rate as reported by the receiver
b_{max}	bound on the rate increase
Δ_b	rate increase factor
Δ_{b_P}	partial rate increase factor

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Berkeley, den 10.2.2000 Jörg Widmer