Improving Congestion Control in IP-based Networks by Information Sharing

vorgelegt von
Diplom-Informatiker
Michael Savorić
aus Berlin

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzender:	Prof. Dr.-Ing. Dr.rer.nat. Holger Boche
Gutachter:	Prof. Dr.-Ing. Adam Wolisz
Gutachter:	Prof. Dr.-Ing. Phuoc Tran-Gia

Tag der wissenschaftlichen Aussprache: 18. Oktober 2004

Berlin 2004

D 83

The first principle is that you must not fool yourself–and you are the easiest person to fool. So you have to be very careful about that. After you've not fooled yourself, it's easy not to fool other scientists. You just have to be honest in a conventional way after that.

Richard Feynman, Cargo Cult Science, 1974

Acknowledgments

First of all, I like to thank Prof. Dr. Adam Wolisz and Prof. Dr. Phuoc Tran-Gia for the acceptance of the research topic of my dissertation and for the review of my dissertation.

During the last six years I have worked as a research assistant in the Telecommunication Networks Group managed by Prof. Dr. Adam Wolisz. I am very grateful to him and to all members of the Telecommunication Networks Group for this very interesting, instructive, and valuable period of time. I will remember it with pleasure.

My dissertation could not be written without the help of many people. From the Telecommunication Networks Group, I specially thank Prof. Dr. Adam Wolisz, Prof. Dr. Holger Karl, and Dr. Morten Schläger for very interesting and productive discussions about my research work. Also the office and technical staff of the Telecommunication Networks Group helped me a lot. Several students and diploma candidates made important contributions to my dissertation and research work: Cornelius Mahlo wrote some of the core parts of the ns-2 simulation framework; Jewgenij Belopilski implemented and evaluated the Linux EFCM controller; Burkhard Daniel, Priscilla Nkweti, Giscard Wepiwe, Adel al-Hezmi, and Daniel Loose implemented and evaluated an earlier version of the FS-TCBI controller in Linux (their work is not explicitly included in my dissertation, but it led to a better understanding of the behavior of FS-TCBI and related controllers in the Internet environment). I thank all of them. From Ericsson Research, Germany, I thank Joachim Sachs, Stephan Baucke, and Dr. Reiner Ludwig for helpful comments about my research activities. In addition, Ericsson Research, Germany, financed most of the research considered in my dissertation.

My parents Lydia and Milan, my twin brother Stefan, my other relatives, and my friends from the chess club CFC Hertha 06 gave me a lot of support during my stay in Berlin and especially during the period of writing my dissertation. For this, I thank them so much.

Finally, I like to thank Prof. Dr. Udo Krieger who aroused my interest in doing research. In addition, he encouraged me to continue my research activities after the study.

Berlin, October 2004

Michael Savorić

Contents

List of Figures

List of Tables

Acronyms

ABR	Available Bit Rate
AECN	AntiECN
AIMD	Additive Increase Multiplicative Decrease
ALF	Application Level Framing
AN	Application Notification
API	Application Programming Interface
AQM	Active Queue Management
ARQ	Automatic Repeat Request
ARWND	Advertised Receiver Window
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BSD	Berkeley Software Design
CA	Congestion Avoidance
CBR	Constant Bit Rate
CF	Congestion Feedback
CH	Congestion Header
CIDR	Classless Inter-Domain Routing
CM	Congestion Manager
CMP	Congestion-Management Proxy
CM-PEP	Congestion-Management PEP
CN	Core Network

COG Center-of-Gravity

CRC Cyclical Redundancy Check

CSFQ Core-Stateless Fair Queueing

CWND Congestion Window

DCM Distributed Congestion Management

DS Differentiated Services

DSL Digital Subscriber Line

EAC Ensemble Ack Clocking

EC Efficiency Controller

ECB Ensemble Control Block

ECN Explicit Congestion Notification

EF Explicit Feedback

EFCM Ensemble Flow Congestion Management

E-TCP Ensemble-TCP

ETCP Enhanced TCP

EWA Explicit Window Adaptation

FBA-TCP Fair Bandwidth Allocation for TCP

FC Fairness Controller

FCC Fuzzy Congestion Controller

FERM Fuzzy Explicit Rate Marking

FERMA FERM Adaptation

FERMAM FERMA Modification

FEWA Fuzzy Explicit Window Adaptation

FLC Fuzzy Logic Controller

FRED Fuzzy RED

FTP File Transfer Protocol

FS-TCBI Fair Share TCBI

FXCP	Fuzzy Explicit Control Protocol
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
IAT	Inter-Arrival Time
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IS	Information Sharing
ISP	Internet Service Provider
I-TCP	Indirect-TCP
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
IW	Initial Window
LAN	Local Area Network
MAC	Medium Access Control
MEP	Mobility-Enabling Proxy
MSS	Maximum Segment Size
NAT	Network Address Translation
NIS	Network-Information Sharing
NLA-ID	Next-Level Aggregation Identifier
PC	Personal Computer
PDU	Protocol Data Unit
PEP	Performance-Enhancing Proxy

PLR Packet Loss Rate

QoS Quality-of-Service

QSR TCP Quick-Start Request

QS-TCP TCP Quick-Start

RAN Radio Access Network

RC Radio Cell

RCF Router Congestion Feedback

RED Random Early Detection

ReSoA Remote Socket Architecture

RLF Router Load Feedback

RNC Radio Network Controller

RTO Retransmission Timeout Timer

RTT Round Trip Time

RTTVAR RTT Variance

SLA-ID Site-Level Aggregation Identifier

SPAND Shared Passive Network Performance Discovery

SRTT Smoothed RTT

SS Slow Start

SSTHRESH Slow Start Threshold

TCB TCP Control Block

TCBI TCP Control Block Interdependence

T/TCP Transactional TCP

TCP Transmission Control Protocol

TFRC TCP-Friendly Rate Control

TLA-ID Top-Level Aggregation Identifier

TOP Timeout Period

TOS Type-of-Service

TTL	Time-To-Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VPN	Virtual Private Network
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network
WWW	World Wide Web
XCP	Explicit Control Protocol

Zusammenfassung

Im heutigen Internet wird der größte Anteil des gesamten auftretenden Datenverkehrs mit Hilfe des Transmission Control Protocol (TCP) übertragen, z.B. WWW- oder FTP-basierter Datenverkehr. TCP besitzt eine eingebaute Fluß- und Überlastkontrolle, um die Datenmenge, das ein einzelner TCP-Sender momentan in das Netzwerk einspeist, zu regeln. Der Zweck des ersten Kontrollverfahrens, der Flußkontrolle, ist es, den TCP-Empfänger vor einer möglichenÜberlast zu schützen. Das zweite Kontrollverfahren, die Überlastkontrolle, versucht einerseits die Router im Netzwerk vor einer Überlast zu schützen, andererseits aber auch eine effiziente Auslastung des Netzwerks zu gewährleisten. Beide Kontrollverfahren sind Fenster-basiert und arbeiten Ende-zu-Ende.

Die Flußkontrolle von TCP arbeitet nach einem sehr einfachen Schema in welchem der TCP-Empfänger einer TCP-Verbindung seinem TCP-Sender die momentan annehmbare Datenmenge durch das Senden eines Flußkontrollfensters innerhalb eine TCP-Quittung mitteilt. Die TCP-Überlastkontrolle ist deutlich komplexer. Der TCP-Sender einer jeden TCP-Verbindung sammelt Informationen über die momentane Lastsituation im Netz. Anhand dieser Informationen wird die aktuell verfügbare Bandbreite im Netzwerk geschätzt und ausgetestet. Dies geschieht durch das Senden von TCP-Segmenten und das Warten auf ihre Bestätigungen, den sogenannten TCP-Quittungen. Aus Sicht eines TCP-Senders bedeutet die Ankunft einer TCP-Quittung, daß das Netzwerk nicht überlastet ist und eine (leichte) additive Vergößerung des Überlastkontrollfensters vorgenommen werden kann. Falls aber eine TCP-Quittung innerhalb einer gewissen Zeitdauer nicht beim TCP-Sender eintrifft, dann vermutet der TCP-Sender eine Überlast im Netzwerk und reagiert darauf mit einer (deutlichen) multiplikativen Reduktion seines Überlastkontrollfensters zur Verringerung der Last im Netzwerk. D.h. jeder TCP-Sender testet die verfügbare Bandbreite des Netzwerks kontinuierlich aus, indem er sein Überlastkontrollfenster — falls möglich — erhöht und sein Überlastkontrollfenster — falls notwendig — reduziert. Obwohl beide Kontrollmechanismen getrennt voneinander durchgeführt werden, werden ihre Resultate miteinander kombiniert, um beiden Kontrollzielen gerecht zu werden. Das eigentliche Sendefenster eines TCP-Senders ist daher das Minimum aus seinem Fluß- und seinem Überlastkontrollfenster.

Es ist offensichtlich, daß eine Überlastkontrolle basierend auf Schätzen und Austesten der momentan verfügbaren Bandbreite des Netzwerks weit von einer optimalen Lösung hinsichtlich der Auslastung des Netzwerks bei gleichzeitiger Verhinderung von Überlast im Netzwerk entfernt ist. Diese Problematik wird in zukünftigen IP-basierten Netzwerken, in denen verschiedenartige drahtgebundene sowie drahtlose Zugangstechnologien nebeneinander genutzt werden, noch verstärkt. Beispielsweise wird dort die Grundannahme der TCP-Überlastkontrolle, daß ausbleibende TCP-Quittungen auf eine Überlast im Netzwerk hindeuten, nicht mehr aufrechterhalten werden können, da Segmente und Quit-

tungen nun nicht nur in überlasteten Routern verloren gehen können, sondern auch infolge von schlechten Übertragungsqualitäten in drahtlosen Kommunikationskanälen verloren gehen (oder zumindest sehr stark verzögert werden) können (da lokale Fehlerkorrekturverfahren aktiviert werden).

Um die Performanz der TCP-Überlastkontrolle zu verbessern, sind einige rudimentäre implizite oder explizite Verfahren der Überlast-Rückmeldungen von Routern entwickelt und teilweise im Internet eingesetzt worden, z.B. Random Early Detection (RED) oder Explicit Congestion Notification (ECN). Aber im allgemeinen sind diese beiden Verfahren nicht in der Lage, die Performanz von TCP deutlich zu verbessern. Hinzu kommt noch, daß diese beiden Verfahren den negativen Einfluß von drahtlosen Kommunikationskanälen auf die TCP-Performanz überhaupt nicht beheben können. Um die TCP-Performanz in heutigen und zukünftigen IP-basierten Netzen beibehalten oder sgar noch steigern zu können, ist daher eine sehr viel durchdachtere TCP-Überlastkontrolle vonnöten.

Im Mittelpunkt dieser Dissertation steht eine Verbesserung der TCP-Performanz durch eine veränderte TCP-Überlastkontrolle, die auf einem Austausch von über das Netzwerk verteilten Informationen basiert. Solch ein verteiltes Überlastkontrollmanagement (Distributed Congestion Management (DCM)) basierend auf dem Austausch von Netzwerkinformationen (Network-Information Sharing (NIS)) besitzt zwei Aspekte. Erstens ist das Wissen über die aktuelle Last im Netzwerk lokal bekannt, z.B. besitzt jeder TCP-Sender sein eigenes Wissen über den aktuellen Netzwerkstatus, und jeder Router ist in der Lage, seine momentane Last zu erfassen. Und zweitens müssen einige dieser lokal verfügbaren Netzwerkinformationen an die TCP-Sender geleitet werden, damit diese ihre Überlastkontrollfenster adequater anpassen und Nutzen daraus ziehen können. Daher bedeutet NIS, daß Netzwerkinformationen im Netzwerk verteilt sind und über das Netzwerk verteilt werden.

Der Austausch von Netzwerkinformationen kann auf zweierlei Arten geschehen: (1) Netzwerkinformationen können zwischen einigen TCP-Sendern innerhalb eines einzelnen Endsystems ausgetauscht werden, und (2) Netzwerkinformationen können zwischen Routern und den TCP-Instanzen in den Endsystemen, deren TCP-Segmente diese Router durchqueren, ausgetauscht werden. Beide Arten können auch in einem hybriden Verfahren miteinander kombiniert werden.

Im Rahmen dieser Dissertation werden bereits existierende und neu entwickelte NIS-Verfahren für TCP hinsichtlich ihrer Funktionalitäten, Eigenschaften, Anforderungen, ihrer Anwendbarkeit im heutigen Internet und ihres zu erwartenden Performanzgewinns im Vergleich zu Standard-TCP klassifiziert und bewertet. Im Mittelpunkt dieser Betrachtungen stehen dabei drahtgebundene IP-basierte Netzwerke; der Einfluß von drahtlosen und mobilen Netzwerken auf die Anwendbarkeit und die Performanz dieser NIS-Verfahren wird aber auch berücksichtigt. Mit Hilfe von analytischen Methoden, Simulationen und Messungen wird gezeigt, daß sorgfältig entwickelte NIS-Verfahren in der Lage sind, die TCP-Performanz deutlich zu steigern, ohne eine nennenswerte zusätzliche Komplexität in den Endsystemen und/oder in den Routern einzuführen.

Zwei wesentliche Ergebnisse werden in dieser Dissertation aufgezeigt. Das erste Ergebnis bezieht sich auf ein neues NIS-Verfahren der ersten Art, genannt Ensemble Flow Congestion Management (EFCM). EFCM ist in der Lage, die Performanz von EFCM-kontrollierten TCP-Verbindungen eines Endsystems deutlich zu verbessern. Verglichen mit Standard-TCP wird dieser Performanzgewinn erreicht, ohne die Aggressivität der TCP-Verbindungen bezüglich der Anzahl ihrer in das Netzwerk eingespeisten Segmente zu erhöhen. Dieses Ergebnis wird dadurch erzielt, daß EFCM den für den Durchsatz und für die Fairness theoretisch besten aber in der Realität kaum erreichbaren Fall von Standard-TCP nachbildet. Das zweite Ergebnis ist verbunden mit neuen NIS-Verfahren der zweiten Art. Diese neuen Verfahren, Fuzzy Explicit Window Adaptation (FEWA) und insbesondere Enhanced TCP (ETCP) basierend auf FEWA, sind in der Lage, den Durchsatz von allen TCP-Verbindungen, deren Segmente einen potentiell überlasteten Router durchqueren, zu erhöhen. Berücksichtigt man neben dem Performanzgewinn auch die Anwendbarkeit der neuen NIS-Verfahren, dann sollten FEWA und ETCP im Internet eingesetzt werden. Als Ergebnis erhält man eine verbesserte Performanz des Netzwerks infolge einer besseren Auslastung des Netzwerks kombiniert mit einer deutlich reduzierten oder sogar komplett verhinderten Überlastung des Netzwerks.

Summary

In the current Internet most of the traffic is transferred by using the Transmission Control Protocol (TCP), e.g., WWW- and FTP-based traffic. TCP has a built-in flow and congestion control to control the amount of traffic which is injected into the network by a single TCP sender. The purpose of the first control mechanism, flow control, is to prevent an overload of the receiver while the aim of the second control mechanism, congestion control, is to prevent overload in the routers of the network combined with an efficient utilization of the currently available bandwidth in the network. Both control mechanisms are window-based and work on an end-to-end basis.

The flow control of TCP follows a very simple scheme in which the TCP receiver of a TCP connection informs the TCP sender about the amount of data it is currently able to accept by sending an advertised receiver window in every acknowledgment back to the sender. TCP's congestion control is much more complex. Here, the TCP sender of a TCP connection collects information about the current load conditions in the network, estimates and probes the currently available bandwidth in the network on the basis of this information. This is done by sending TCP segments and waiting for acknowledgments for these segments. From the TCP sender's perspective, a received acknowledgment indicates non-congestion in the network so that the congestion window can be (slightly) additively increased to improve the network utilization. But if a TCP segment is not acknowledged during a specific time, TCP assumes congestion in the network. Then, the TCP sender has to (largely) multiplicatively decrease its congestion window to reduce the load in the network. Thus, TCP continuously probes the available network bandwidth by increasing its sending window, if possible, and by decreasing its sending window, if necessary. Although both control mechanisms of TCP work separately from each other their results are combined to determine the current sending window of a TCP sender as the minimum of the current advertised receiver window and the current congestion window. This is done to reach the aims of both control mechanisms.

It it obvious that a congestion-control mechanism based on estimating and probing the available network bandwidth is far from optimality in terms of utilizing the network and preventing congestion in the network. This problem will be strengthen in future IP-based networks where different wired and wireless access technologies will be used. For example, the assumption of TCP that missing acknowledgments indicate congestion cannot be retained any more because then segments and acknowledgments cannot only be lost in congested routers, they can be also lost (or at least largely delayed) in wireless links due to bad transmission qualities in these wireless links (that existing local repair mechanisms based on retransmissions are activated).

In order to improve the performance of TCP's congestion control, some rudimentary implicit or ex-

plicit mechanisms for congestion feedback from routers, e.g., Random Early Detection (RED) or Explicit Congestion Notification (ECN), have been developed and implemented in the current Internet. But in general, these mechanisms cannot largely improve the performance of the basic congestion control of TCP. In addition, these two mechanisms do not take into account the negative effects of wireless networks for the performance of TCP. Thus, a more sophisticated TCP congestion control is needed to keep and even improve the performance of TCP in current and future IP-based networks.

The focus of this dissertation is on improving the performance of TCP's congestion control currently used in the Internet by taking advantage of sharing network information which is distributed over the network. Such a distributed congestion management (DCM) based on network-information sharing (NIS) has two aspects. First, information about the current load in the network is locally known, e.g., each TCP sender has its own perspective on the current network status and each router is able to detect its current load. And second, some of this locally available network information has to be transferred to the TCP senders that they can benefit from it by more appropriately setting their congestion and sending window. So, NIS means that network information is distributed in and has to be distributed over the network.

A network-information sharing can be done in two different ways: (1) Network information can be shared in a single sending end system between senders of some TCP connections, and (2) network information can be shared between routers and TCP instances running in end systems whose TCP connections traverse these routers. It is also possible to combine both NIS types in a hybrid approach.

This dissertation classifies and evaluates existing and newly developed NIS approaches for TCP by considering both types of NIS approaches according to their functionalities, properties, complexity, applicability in the current Internet, and expected performance gain compared to standard TCP. The main focus of this dissertation is on NIS approaches for TCP in wired IP-based networks. But the applicability and expected performance gain of NIS approaches for TCP in combined wired, wireless, and mobile IP-based networks are also considered. Analytical methods, simulations and measurements are performed to show that well-developed NIS mechanisms are able to largely improve TCP's performance without introducing considerably additional complexity in end systems and/or in routers.

Two main results are established in this dissertation: The first result is that one of the new NIS approaches of the first type, called Ensemble Flow Congestion Management (EFCM), is able to largely improve the performance of EFCM-controlled TCP connections of a single end system. And compared to standard TCP, this performance gain of EFCM is reached without increasing the aggressiveness considering the number of segments injected into the network. In more detail, EFCM reaches a nearly optimal aggressiveness, i.e., it emulates the theoretical but in reality hardly reached best case of standard TCP regarding the fairness and throughput. The second result is related to NIS approaches of the second type. The new approaches Fuzzy Explicit Window Adaptation (FEWA) and especially Enhanced TCP (ETCP) based on FEWA are able to (largely) improve the performance of all TCP connections traversing a bottleneck router. If both the performance and applicability of the different new NIS approaches is taking into account, FEWA and ETCP should be deployed in the Internet to reach an improved performance of the network due to a better utilization combined with a reduced or even prevented overload in the network.

Chapter 1

Introduction

1.1 Packet-Oriented Networks

Packet-oriented networks like the Internet transfer data between end systems in units of fixed or variable but limited size, the so-called packets. The intermediate network nodes of such networks, the so-called routers, are responsible to forward arriving packets on their ways from their sources to their destinations. Each router is equipped with a buffer of limited size, the so-called queue, to temporarily store packets that cannot be immediately forwarded.

It is known from the queueing-systems theory (cf. [1]) that if the packet-arrival rate is larger than the packet-forwarding or packet-service rate of the router for a longer duration, the queue length of the router grows up to its maximum. Then, additional arriving packets cannot be temporarily stored in the filled queue and have to be discarded by the router. Such a router is called congested. It is obvious that due to packet losses congestion in (parts of) such a network is detrimental for the overall performance of (parts of) the network regarding the efficiency of the data transmission and the utilization of (parts of) the network. Since packet-oriented networks have the inherent property that they can be (locally) congested, congestion control has to be performed in such networks to improve the overall network performance by controlling the load in the network. Here, controlling the load in the network does not mean to restrict the number of concurrent data streams in the network as it is done by using an access control for new data streams entering the network. Congestion control is used to control the load produced by all data streams in the network. This is done by adapting the sending rate of each source of the data streams on the current load conditions in the network. Such a sending-rate adaptation of senders has to be done efficiently to reduce or even prevent congestion but also to allow a high utilization of the currently available bandwidth in the network.

1.2 Congestion Control in Packet-Oriented Networks

In packet-oriented networks, two fundamental types of congestion-control mechanisms can be distinguished regarding the role of the network protocol:

(1) Congestion-control mechanisms can be included in and supported—or at least assisted—by the

network protocol and the routers of the network. Sources are frequently informed by the network protocol about the current load in the network. As a result, network information collected by a source and stored in its congestion-control variables reflects the current load conditions in the network. Then, each source is able to perform an adequate congestion control that both a high utilization of the network and a large performance of the data streams can be expected. This advantage of such a congestion-control mechanism is combined with two disadvantages. First, the congestion-control information transferred by the network protocol requires some additional overhead. In more detail, there is a trade-off between the frequency / overhead and the benefit that can be expected if such a congestion-control mechanism is performed. And second, the upper-layer protocols working on top of the network protocol are limited in their flexibility, since they have to evaluate and react on the congestion-control information supported by the network protocol.

(2) Congestion control can be excluded from and not supported by the network protocol and the routers of the network. Then, protocols working on top of the network protocol are responsible for the congestion control in the network. In this case, each source has to frequently collect network information, stores them in its congestion-control variables, and locally performs congestion control based on these variables. One main problem of this approach is that the network information collected by a sender does not reflect very well the current network conditions. The result is a suboptimal congestion control in terms of network utilization and data stream performance. Another problem of this approach is that the source of each new data stream entering the network does not know anything about the current load conditions in the network. Therefore, such a source starts sending its data very conservatively using a small sending rate, estimates and probes the current network-load conditions by continuously increasing its sending rate. After a while, the TCP sender has raised its local knowledge about the current network load little by little that it is able to perform a more accurate congestion control based on the so far collected network information. In the meantime, the congestion control of this data stream might be also far from optimality.

For example, the first type of congestion-control mechanism is used as the basic standardized congestion control for the Available Bit Rate (ABR) service class that supports best-effort data streams in Asynchronous Transfer Mode (ATM) networks [2, 3, 4, 5, 6, 7]. The second type of congestion-control mechanism is used for data streams of applications using the Transmission Control Protocol (TCP) in the Internet [8, 9, 10, 11, 12].

Possible solutions for a congestion control in packet-oriented networks are generally discussed in [13]. The properties, advantages, and disadvantages of, for example, window- / rate-based, open / closed loop, and router- / source-based congestion-control mechanisms are compared. The main outcome of this comparison is that each of these mechanisms cannot guarantee a well-performing congestion control in a network in all cases. Dependent on the (typical) duration of the overload observed in a network, a combination of several congestion-control mechanisms is necessary to establish an efficient congestion control in the network.

1.3 Congestion Control in the Internet

As already mentioned, congestion control in the Internet is not part of its network protocol IP, it has to be performed by upper-layer protocols, e.g., by the widely used transport-layer protocol TCP. The main features and properties of TCP's basic congestion-control mechanisms are briefly explained in the following: TCP's congestion control is window-based and performed end-to-end without any support of the network. Information about the current conditions in the network path from the source (the TCP sender) to the destination (the TCP receiver) of a TCP connection is locally elaborated and stored in the TCP sender. This network information is used to estimate and probe the currently available bandwidth in the network path by adapting the sending window. This is done by combining the built-in error-control functionalities of TCP with congestion-control mechanisms described in the following. A TCP sender starts to send segments by using a relative low conservatively chosen initial sending window and waits for the acknowledgments of the sent segments. From the TCP sender's point of view, the reception of an acknowledgment is used as an indicator for a non-congested network path. In this case, the sending window is slightly additively increased (AI). On the other hand, if no acknowledgment has been received in a given time interval until the retransmission timeout timer has been expired, i.e., either the TCP segment or its acknowledgment has been lost, the network path seems to be congested. In this case, the sending window is largely multiplicatively decreased (MD). Thus, TCP's congestion control is based on the widely used AIMD mechanism. Its overall performance is therefore vulnerable to segment losses.

This basic TCP congestion control can be assisted by some rudimentary implicit or explicit congestion-control feedback from the routers of the network. Examples are Random Early Detection (RED) [14] or Explicit Congestion Notification (ECN) [15]. Since these two approaches provide only a single-bit network information (no congestion / congestion) as feedback from the routers, their performance gain is rather limited.

1.4 Advanced Congestion Control in the Internet based on Network-Information Sharing

Due to the more or less accurate network information collected by a TCP sender during the lifetime of its TCP connections, even in today's Internet, the performance of the TCP congestion control possibly assisted by some simple router congestion feedback (RCF) approaches is very often far from optimality regarding the throughput of a TCP connection and the utilization of the network path. This problem will be strengthen in future IP-based networks, where the transparent integration of different wired and wireless access technologies with specific, (very) different, and dynamic bandwidth, delay, and error characteristics will play an important role. In order to improve the overall performance of the current and future Internet, it might be advantageous to assist TCP's end-to-end congestion control with new congestion-control functionalities based on sharing network information distributed over the network Two different types of network-information sharing (NIS) approaches can be distinguished:

(1) Network information can be shared between some TCP senders of a single end system, or

(2) network information can be shared between routers and TCP senders located in end systems.

Every TCP sender collects its own network information and performs its congestion control separately from all other TCP senders of the end system. For some of these TCP connections, i.e., for those which traverse the same network path or at least the same bottleneck link, a NIS approach of the first type might be beneficial for the performance, since then each of these TCP senders might get more up-to-date and more accurate network information than in the separately controlled case.

In general, each router in the Internet is able to measure its current load. If a NIS approach of the second type is used, the load information of each router can be sent back to the TCP senders of all TCP connections traversing this router. Then, the TCP senders are able to react much faster and more accurate on the current load in the router by adapting their sending window accordingly. The NIS approach of the second type can be understood as a mechanism to equip the Internet with a congestion control of the first type, in the broadest sense related to that used in ATM networks.

Thus, both NIS approaches have the potential to (largely) increase the performance of some or all TCP connections in the Internet. It is therefore worth to consider both NIS approaches in much more detail.

1.5 Contents of the Dissertation

The focus of this dissertation is on the comparison of existing and new NIS approaches of both types with regard to their functionalities, properties, requirements, additional complexity, applicability in the current Internet and in future IP-based networks, and expected performance gain compared to standard TCP. Some of these NIS approaches are then selected for a detailed performance evaluation by simulations and—in parts—also by measurements. In addition, a mathematical analysis of one of the new NIS approaches of the first type is performed to investigate its main congestion-control algorithms also from a theoretical point of view.

1.6 Structure of the Dissertation

In Chapter 2, a short overview about the network protocols IP and IPv6 used in the current Internet is given. In addition, the main properties and functionalities of the transport protocols TCP and UDP are shown. This includes a detailed description of the error-, flow-, and congestion-control functionalities of TCP and the definition of metrics used to evaluate the performance of TCP. And Chapter 3 gives a general overview about the two possible fundamental types of NIS approaches in the Internet. The rest of this dissertation is separated in two parts.

In the first part, Part I, NIS approaches in a sending Internet end system are considered. This part of the dissertation consists of the Chapters 4 to 11. In Chapter 4, existing NIS approaches for sending Internet end systems are described regarding their functionalities, properties, requirements, additional complexity, and expected performance gain compared to standard TCP. The algorithms and features of two new NIS approaches for sending Internet end systems called Fair Share TCBI (FS-TCBI) and Ensemble Flow Congestion Management (EFCM) are shown in Chapter 5. These two NIS approaches are considered in more detail in the rest of the first part of the dissertation. In Chapter 6, the congestion-control algorithms of standard TCP and EFCM are analytically compared to show that both approaches have a similar aggressiveness to put segments into the network. The performance metrics used to evaluate the performance of the new NIS approaches compared to standard TCP are given in Chapter 7. In addition, the simulation model used to investigate the performance of these two NIS approaches compared to standard TCP is described in the same chapter. Simulation results for FS-TCBI can be found in Chapter 8 while simulation and measurement results for EFCM can be found in Chapter 9 and 10, respectively. Chapter 11 concludes the first part of this dissertation and gives an outlook to future research activities with regard to the EFCM approach and further interesting aspects of NIS approaches for sending Internet end systems.

The second part of this dissertation, Part II, considers NIS approaches between Internet routers and end systems. This part consists of the Chapters 12 to 18. Related work of these NIS approaches can be found in Chapter 12. Two new such NIS approaches called Fuzzy Explicit Window Adaptation (FEWA) and Enhanced TCP (ETCP) are described in Chapter 13. A comparison of the main functionalities, properties, requirements, additional complexity, and expected performance gain compared to standard TCP of all these NIS approaches including their applicability in the current Internet is given in Chapter 14. The two new NIS approaches and the existing NIS approaches EWA and Explicit Control Protocol (XCP) are considered in more detail in the rest of the second part of the dissertation. In Chapter 15, the performance metrics used to evaluate the performance of these NIS approaches are given. In addition, the simulation model used to investigate the performance of these four NIS approaches compared to standard TCP is described. Simulation results for EWA and FEWA can be found in Chapter 16 while simulation results for ETCP and XCP can be found in Chapter 17. Chapter 18 concludes the second part of this dissertation and gives an outlook to future research activities with regard to FEWA, ETCP, and XCP and their usage in fixed, wireless, and mobile IP-based networks.

Chapter 19 concludes this dissertation and gives and outlook to future research activities in the field of NIS approaches in current and future IP-based networks.

Appendices A and B of this dissertation contain the results of transient analysis and of the statistical evaluation of the simulations performed in Part I of this dissertation. For Part II, these results are given in Appendices C and D. In addition, the parameters of the new NIS approach FEWA are described in detail in Appendix E. Finally, for all NIS approaches between routers and Internet end systems considered in this dissertation, their parameters and typical values are summarized in Appendix F.

Chapter 2

The Internet

2.1 Introduction

The Internet is a packet-oriented network. It provides an unreliable, connection-less, and best-effort transport of packets between Internet end systems and routers using a network protocol, called Internet Protocol (IP). IP specifies the packet format, performs the routing of packets through the Internet, defines rules how Internet end systems and routers should process or discard packets and how and when error messages should be generated by end systems or routers. Therefore, IP is the fundamental part of the Internet [12]. A group of hosts or whole networks that want to be part of the Internet have to support at least this protocol to be able to communicate with other end systems and routers of the Internet. Then it is possible to connect heterogeneous networks based on different underlying technologies to the Internet.

On top of IP, several transport protocols are standardized that provide additional functionalities for the applications running in the end systems than IP does, for example, a reliable transport of packets through the Internet. Figure 2.1 shows the protocol stack of the Internet together with typically protocols used in the Internet Environment. In the application layer, these protocols are, for example, Hypertext

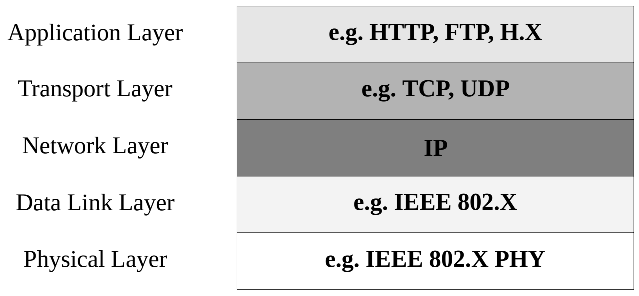


Figure 2.1: Typical protocols in the Internet environment

Transfer Protocol (HTTP) [16, 17, 18] to exchange World Wide Web (WWW) pages, File Transfer Protocol (FTP) [19] to transfer files, or some protocols of the ITU-T H.X protocol family to send video or audio streams, e.g., H.323 to support Internet telephony [20]. The transport-layer protocols Transmission Control Protocol (TCP) [9] or User Datagram Protocol (UDP) [21] are used to partion the data of these application-layer protocols into segments and transmit them between the end systems of the Internet on an end-to-end basis. The network-layer protocol IP [8] is used to encapsulate the segments of these transport-layer protocols into packets and transmit them through the Internet on a hop-by-hop basis.

Protocols in layers below the network layer are working link-wise between a pair or a group of network nodes, i.e., Internet routers and end systems, to transfer information. For local area networks (LANs), for example, these protocols are part of the standardized IEEE 802.X protocol family, e.g., Ethernet (802.3) [22, 23] or wireless LAN (WLAN) (802.11) [24] with their specific data-link layer and physical-layer protocols. Figure 2.2 shows how a typical application for the Internet environment like WWW can be expressed in terms of the Internet protocol stack.

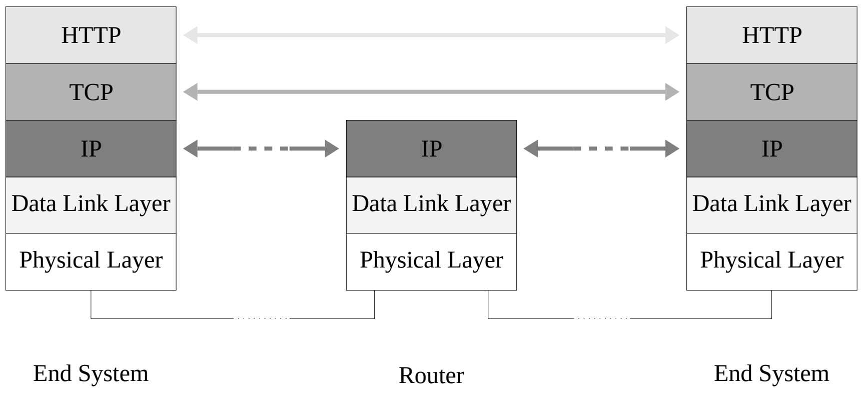


Figure 2.2: A single TCP connection in the Internet protocol stack

The data stream generated by an application, e.g., the WWW page requested by a remote browser, is segmented in pieces of limited size. Each of these WWW-based data-stream pieces is put as payload together with a TCP header into a single TCP segment (in the context of TCP PDUs are called segments). An IP PDU, also called IP packet, consists of an IP header and its payload carrying, in this case, a single TCP segment. Thus, the WWW-based data stream is segmented and encapsulated in TCP and IP PDUs to be transferred through the Internet. This is shown in Figure 2.3.

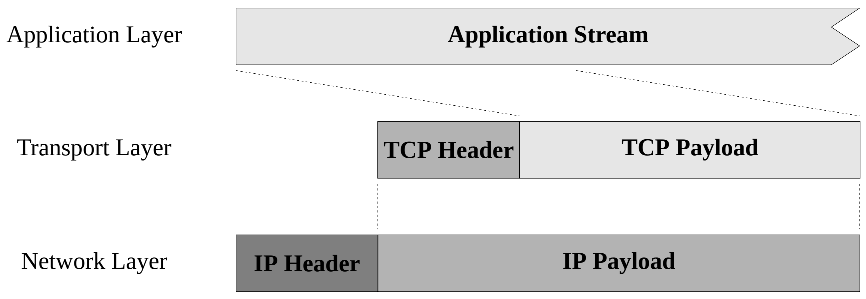


Figure 2.3: Encapsulating of data of a typical Internet application like WWW into TCP/IP PDUs

The remainder of this chapter is organized as follows. Section 2.2 briefly describes the main properties of IP regarding its two currently implemented versions 4 and 6. And Section 2.3 contains the detailed description of the two most important transport protocols TCP and UDP currently used in the Internet environment.

2.2 Internet Protocol

2.2.1 Introduction

In the current Internet IP in its version 4 [8] is mainly deployed. Since this IP version is used over more than twenty years, IPv4 and IP are used as synonyms. But a following IP version 6, called IPv6 to distinguish it from IP(v4), has been already specified [25, 26] and implemented in several routers and end systems. Since IPv6 is not widely deployed in the current Internet environment, it is only marginally considered in this dissertation.

Both IP versions provide an unreliable transfer of packets between and systems and routers. These packets can contain data from upper layer protocols running on end systems but also data of IP-supported protocols, e.g., network-control information based on the standardized Internet Control Message Protocol (ICMP) to indicate errors or other important events in the Internet. In addition, the IP versions are able to support not only communication between a pair of end systems (unicast) but also between groups of end systems (multicast). Throughout this dissertation, only unicast communication is considered. The sending end system of an IP packet is called an IP sender while the receiving end system of an IP packet is called an IP receiver.

This section contains a detailed description of the structure of an IP PDU as well as of an IPv6 PDU. Additional properties of these two IP versions, e.g., their address structure for identifying end systems in the Internet, are mentioned insofar as they are necessary with regard to the later described NIS approaches in the Internet environment.

2.2.2 IP PDU

In Figure 2.4 the PDU of IP, called IP packet, is shown. It consists of the header and the payload. The

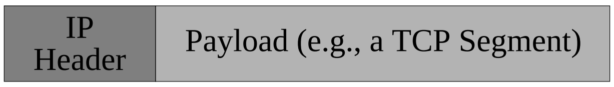


Figure 2.4: IP packet

IP header is shown in Figure 2.5. In the following, the functionalities and possible values of each field in the IP header of an IP packet are described: The first field of the IP header, the version field, contains the version of the used IP. It is set to 4 to reflect that this header is used in the fourth IP version. By reading this field, routers and end systems are able to distinguish between different IP versions. Due to selectable options in the IP header, the length of the IP header can vary. Therefore, an IP header must contain its length which is stored in the header-length field (HLen). In this 4-bit long field, the length of the IP header in multiples of 32 bits is stored. Possible values range from 5 (an IP header without any additional options) to 15 (an IP header with at most 40 bytes of options or padding, respectively). The next field of the header is the type-of-service (TOS) field which defines the service type an IP packet belongs to. It was originally specified to allow a distinction between packets of different quality-of-service (QoS) classes in the network. In the current Internet this field is used, for example, to distinguish between

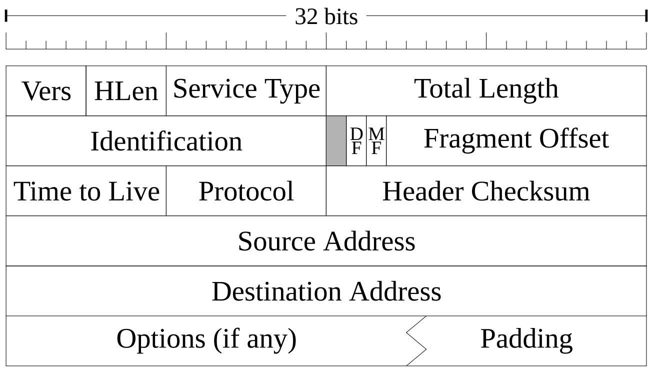


Figure 2.5: IP header

routing-information and other IP packets in a router that a router is able to receive routing information even if it is currently congested. In addition, the service classes of Differentiated Services (DS) are determined by redefining the semantic of this field [27]. But the TOS field can be also used for other mechanisms than it was basically specified for. The total length of the IP packet is stored in the fourth field of the header. Since this field has a size of 16 bits, the maximum allowed length of an IP packet is 65535 ($2^{16} - 1$) bytes. Header-field five contains an identification number that uniquely identifies each IP packet sent by an end system. This identification number is used in the context of fragmented IP packets. The next bit of the IP header is reserved. It could be used in future. Since IP is designed to be used on top of various network platforms, it is possible that some network platforms support only smaller packets than other networks. Thus, it might be sometimes necessary that a router has to fragment a single IP packet of a too large size into several IP-packet fragments of smaller size. The first fragment-control bit in the IP header, called don't-fragment (DF) bit, enables (0) or disables (1) the fragmentation of the IP packet in a router. But if an IP packet is fragmented, the second fragment-control bit, called more-fragments (MF) bit, in the IP header denotes whether an IP-packet fragment is the last (0) fragment of an original IP packet or not (1). The fragment offset is used to determine the order of IP-packet fragments in the original IP packet. Together with the equal identification number and the different fragment offsets stored in each fragmented IP packet of an original IP packet a destination is able to restore the original IP packet from its fragments. The time-to-live (TTL) field in an IP packet is used to limit the lifetime of an IP packet in the network. The source of an IP packet sets this field to an appropriate large value, e.g., 64, and every router receiving an IP packet decreases this field by one. If a router receives an IP packet with a TTL of one, the router discards the IP packet and informs the source of the IP packet about this packet-discard. Thus, the maximum hop-lifetime of any IP packet in the Internet is limited to 255 visited intermediate routers. The next header field contains the protocol identifier of the transport protocol the payload of an IP packet belongs to. Possible values of this field are, for example, 6 for TCP and 17 for UDP. The following header field contains a checksum for the whole IP header. Since some fields in the IP header change their value during the transfer through the network, e.g., the TTL field is decreased by every traversed router, this checksum has to be recalculated in each router. With this checksum, routers and end systems are able to detect IP packets with erroneous IP headers and discard them. The next two header fields contain the source and destination addresses of the IP packet. In IP, each end system has a 32-bit address whose structure is explained in more detail in Section 2.2.3. If options in the IP header are used, they follow directly after the destination address. Since the length of an IP header is oriented in multiples of 32 bits, additional bits have to be used to pad the header if the IP options do not end on such a 32-bit boundary.

2.2.3 IP Addresses

An IP address has a size of 32 bits. It is hierarchically structured in three parts: a prefix used to separate different classes of IP addresses, a network-address part that identifies a whole network in the Internet, and a host-address part that identifies a single end system in a network (more precisely: a network interface in a network node). For unicast communications three different IP-address classes can be distinguished:

- Class A: Such IP addresses start with a 1-bit prefix of 0. The network part is 7 bits long and the host part is 24 bits long. Thus, at most 2^7 class-A networks with at most 2^{24} end systems in each of them can be established in the Internet environment.
- Class B: Such IP addresses start with a 2-bit prefix of 10. The network part is 14 bits long and the host part is 16 bits long. Thus, at most 2^{14} class-B networks with at most 2^{16} end systems in each of them can be established in the Internet environment.
- Class C: Such IP addresses start with a 3-bit prefix of 110. The network part is 21 bits long and the host part is 8 bits long. Thus, at most 2^{21} class-C networks with at most 2^8 end systems in each of them can be established in the Internet environment.

The stated maximum numbers of possible networks of the three different network classes and the maximum numbers of possible end systems in each of these network classes are upper bounds, since some of the IP addresses are reserved for specific communications, e.g., broadcast or loopback, and cannot be used to address single networks and/or end systems.

Inside a single network, e.g., a class-B network, it is possible to locally install subnetworks. This is done by using some bits of the host part of the IP address as subnetwork identifiers. Then, routers inside the single network are able to distinguish between these subnetworks by using the locally modified semantic of the IP-address host part that they can forward packets to the correct destinations in the subnetworks.

The rigid partition of globally used IP addresses in a network and a host part of fixed size can be globally eliminated if Classless Inter-Domain Routing (CIDR) [28] is used. Then, a partition of IP addresses in finer scaled network and host parts is possible that networks with maximum numbers of end systems between 2^5 up to 2^{19} can be installed.

2.2.4 IPv6 PDU

In Figure 2.6 the PDU of IPv6, called IPv6 packet, is shown. It consists of the base header, several optional extension headers, and the payload. The important difference of IPv6 compared to the current

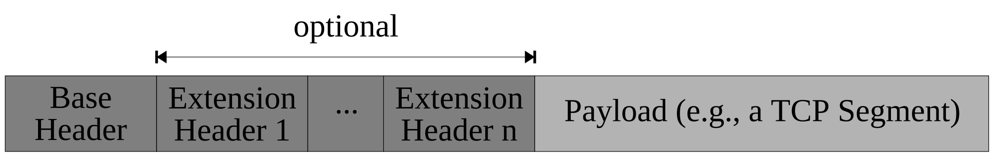


Figure 2.6: IPv6 packet

IP version is that the base IPv6 header has a fixed size and contains only those fields which are essential to transfer packets from one end system to another end system. Other functionalities, e.g., IPv6 options, packet-fragmentation functionalities, or security mechanisms like IPSec [29], are released from the base IPv6 header and placed in several IPv6 extension headers that can be concatenated one after another following the base IPv6 header. With such a header structure a router is able to faster forward packets, since extension headers are only considered by the router if they are necessary to support a specific router-functionality.

The base header of IPv6 is shown in Figure 2.7. In the following, the functionalities and possible

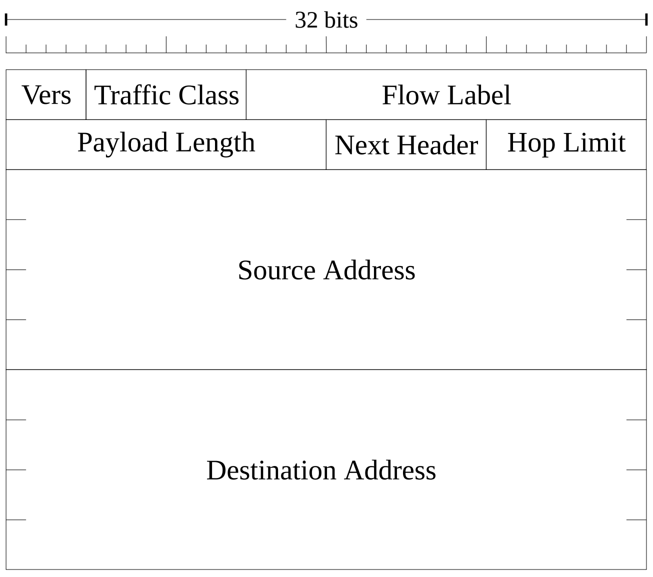


Figure 2.7: IPv6 base header

values of each field in the IPv6 base header of an IPv6 packet are described: Similar to IP, the IPv6 header starts with a version field. Here, this field contains 6 to reflect that this header is used in an IPv6 packet. The next field contains the traffic class of an IPv6 packet. This is analogous to the TOS field in an IP header. The third field in an IPv6 header defines a flow label for the IPv6 packet. This information can be used by routers to identify IPv6 packets of different flows. Field four contains the payload length of an IPv6 packet. Since this field has a size of 16 bits, the maximum allowed payload length of an IPv6 packet is 65535 ($2^{16} - 1$) bytes. The next-header field in an IPv6 header defines the type of the header following the base IPv6 header. This can be the identifier of an extension header or identifiers for upper-layer protocols, e.g., TCP or UDP. In the latter case, the semantic of this field is similar to the protocol field in an IP header. The hop-limit field of an IPv6 header has the same function than the TTL field in an IP header: it counts the number of traversed intermediate routers to limit the maximum hop-lifetime of an IPv6 packet in the network. The last two fields of an IPv6 header contain the source and destination addresses of an IPv6 packet. In IPv6, each end system has a 128-bit address. In IPv6, each end system has a 128-bit address whose structure is explained in more detail in Section 2.2.5.

Functionalities of IPv6 that are not supported by fields in the base IPv6 header can be provided by fields in extension headers. Depending on these functionalities, e.g., IPv6 options, packet fragmentation, or IPSec mechanisms, the extension headers can have a different internal structure. But each extension header consists of at least two fields: the next-header field and a field containing the length of the extension header (the latter field may be omitted if an extension header has a fixed size). Extension headers which do not end on a 64-bit boundary have to use padding information to meet this new requirement of IPv6.

2.2.5 IPv6 Addresses

An IPv6 address has a size of 128 bits. There exist many different types of IPv6 addresses. But in this section, only IPv6 addresses related to the IP-class addresses A to C are briefly described. Such IPv6 addresses, called aggregated unicast IPv6 addresses, consists of a 3-bit prefix of 001, a 61-bit network part and a 64-bit host part. The network part is separated in four fields to establish hierarchically structured networks: a 13-bit top-level aggregation identifier (TLA-ID) to identifier the highest hierarchical level of the network, a 8-bit field reserved for future use, a 24-bit next-level aggregation identifier (NLA-ID) to identifier the next-lower hierarchical level of the network, and a 16-bit site-level aggregation identifier (SLA-ID) to finer partition the network into subnetworks. Typically, TLA-IDs are used to identify international or national-wide operating network providers, NLA-IDs are used to identify regional network providers or organizationally separated parts of a larger network, and SLA-IDs are used by private network carriers, e.g., universities or companies, to partition their networks into subnetworks.

2.3 Transport Protocols in the Internet

2.3.1 Introduction

The most important transport protocols in the current Internet are the Transmission Control Protocol (TCP) [9, 12] and the User Datagram Protocol (UDP) [21, 12]. TCP is a widely used transport protocol in the current Internet. It provides a reliable transport of data. Many well-known application protocols, e.g., HTTP and FTP, are based on TCP. In contrast to TCP, UDP cannot guarantee a reliable transport of data. It is therefore mainly used in the current Internet for the transport of data whose performance is generally more affected by delay than by losses, e.g., real-time audio or video traffic. In this section, the main properties and functionalities of these two transport protocols are described in detail.

2.3.2 Transmission Control Protocol (TCP)

TCP is a connection-oriented transport protocol. Therefore, before any data can be exchanged between the two TCP instances of a TCP connection the TCP connection has to be established. And after the TCP instances have finished their data transfer, the TCP connection has to be closed.

Both TCP instances of a TCP connection are able to simultaneously send and receive payload data in two directions, i.e., TCP is a full-duplex transport protocol. Sometimes, one of the two TCP instances is named TCP sender and the other one is named TCP receiver. This is done to specify more precisely the main (temporary) direction of the data transfer between the two TCP instances. In this dissertation, an end system in which a TCP sender is located is called a sending end system. The end system in which a TCP receiver is located is called a receiving end system.

TCP is a byte-oriented stream protocol. This means that every byte of the data stream transferred in one direction between the TCP instances of a single TCP connection is consecutively numbered and can be identified by its unique sequence number. This allows a TCP receiver to detect whether payload bytes are lost or not during the transfer.

Since the underlying IP protocol is packet-oriented, a TCP sender has to divide the data stream received from an upper-layer protocol in TCP-specific protocol data units (PDU's), called TCP segments, of limited size. Each of these TCP segments is additionally protected by a checksum against corruption that a TCP receiver is able to also detect whether a received TCP segment contains erroneous bytes or not. This feature is used by the TCP receiver to acknowledge the TCP sender the reception of error-free received TCP segments. Then, the TCP sender is able to know which TCP segments have been correctly transferred to the TCP receiver and which TCP segments have to be retransmitted due to losses or errors. This is the built-in error-correction functionality of TCP.

In this section, the structure of a TCP PDU is described, the setup and teardown of a TCP connection is depicted, and it is explained how the error correction, flow, and congestion control of TCP works. In addition, the main performance metrics to evaluate TCP are defined and two mathematical formulas to calculate the mean throughput a single TCP connection can expect are given.

2.3.2.1 TCP PDU

The PDU of TCP, called TCP segment, is shown in Figure 2.8. It consists of a header and the payload

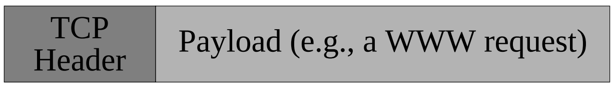


Figure 2.8: TCP segment

carrying, for example, the request of a WWW page. The TCP header is shown in Figure 2.9. In the following, the functionalities and possible values of each field in the TCP header of a TCP segment are described: The first two fields in the TCP header contain the source and destination port of the TCP connection. Together with the source and destination address of the end systems stored in the IP header these two values are used to clearly identify the two processes in the end system which are linked with

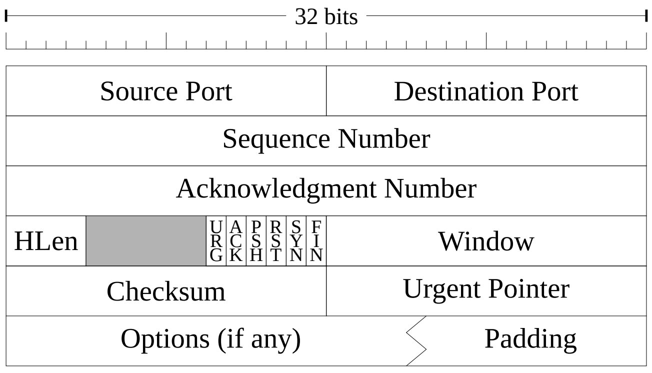


Figure 2.9: TCP header

the TCP connection. Every byte in a TCP stream is numbered. The sequence number (SeqNr) in the TCP header stores the number of the first payload byte in the TCP segment. The acknowledgment number (AckNr) is set to the number of the last acknowledged byte plus one, i.e., it contains the number of the next expected byte in the payload data stream. Due to selectable options in the TCP header, the length of the TCP header can vary. Therefore, a TCP header must contain its length which is stored in the header-length field (HLen). In this 4-bit long field, the length of the TCP header in multiples of 32 bits is stored. Possible values range from 5 (a TCP header without any additional options) to 15 (a TCP header with at most 40 bytes of options or padding, respectively). The next 6 bits are reserved. They could be used in future versions of TCP. The following 6 bits are the so-called code bits, They are used to further specify the type of the payload or the TCP segment. If the urgent bit (URG) is set, the urgent pointer is valid. Then the payload (or parts of the payload) is marked as urgent data. These data should be immediately transferred to the receiver, even if this requires the sending of short TCP segments much smaller than the allowed maximum segment size (MSS). The acknowledgment bit (ACK) is set in a TCP segment to indicate that the AckNr field is valid. Then this segment acknowledges data or confirms the connection-setup or connection-teardown request of a remote TCP instance (cf. Section 2.3.2.2). The push bit (PSH) is used to tell the receiver of the TCP segment to immediately acknowledge this segment after reception. With the reset bit (RSH) the whole TCP connection can be re-initialized by resetting the TCP instances. The SYN or FIN bit are set in the first or last TCP segment of a TCP stream of one direction to setup or teardown the TCP connection in this direction. The window-field is used for TCP's flow control. It contains the number of bytes that the remote TCP instance is currently able to receive. Possible values range from 0 to 2^{16} bytes. Higher values up to 2^{32} bytes are possible if both TCP instances agree on shifted values for these bits during the connection-setup phase by using the TCP window scale option (cf. [30]). Since TCP is a reliable protocol based on Automatic-Repeat Request (ARQ) [22] mechanisms, i.e., the remote TCP instance acknowledges every correctly received TCP segment, it uses a checksum for the whole PDU that the remote TCP instance is able to detect errors in the TCP segment. This checksum is calculated by a Cyclical Redundancy Check (CRC) [22] based on the generator polynomial $X^{16} + X^{15} + X^2 + 1$. The urgent pointer points at the beginning of urgent data in the payload. If options in the TCP header are used, they follow directly after the urgent pointer. Since the length of a TCP header is oriented in multiples of 32 bits, additional bits have to be used to pad the header if the TCP options do not end on such a 32-bit boundary.

2.3.2.2 TCP Connection Setup and Teardown

Before any data can be transferred between the two TCP instances, a TCP connection must be established between them. This TCP connection-setup is performed by the three-way-handshake mechanism shown in Figure 2.10.

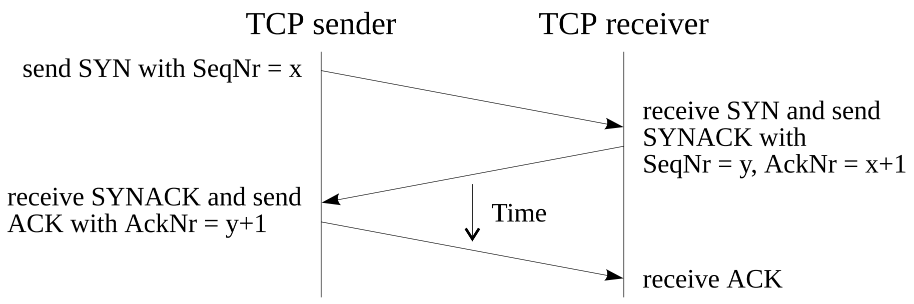

TCP sender
TCP receiver
send SYN with SeqNr = x
receive SYN and send SYNACK with SeqNr = y, AckNr = x+1
receive SYNACK and send ACK with AckNr = y+1
Time
receive ACK

Figure 2.10: Three-way-handshake connection-setup of TCP

In addition, the TCP instances initialize their control variables and are then able to start sending data bytes which are uniquely numbered according to the negotiated initial sequence numbers, one for each direction. After the connection-setup, the TCP instances are able to send their data. If the data transfer has been finished, the TCP connection is closed by a modified three-way handshake mechanism shown in Figure 2.11.

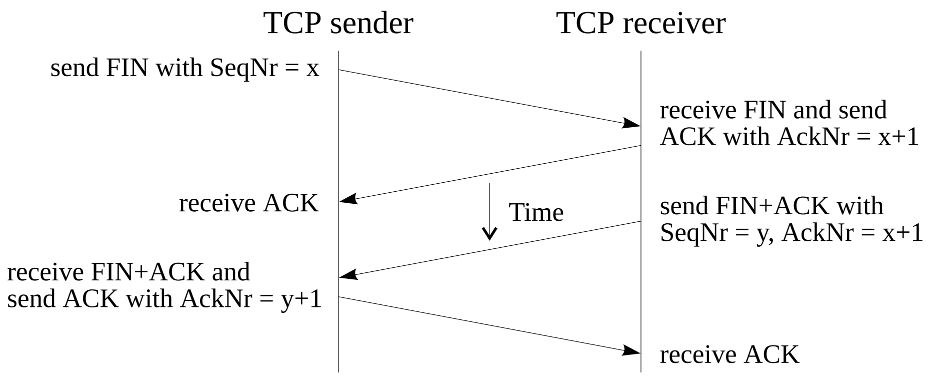

TCP sender
TCP receiver
send FIN with SeqNr = x
receive FIN and send ACK with AckNr = x+1
receive ACK
Time
send FIN+ACK with SeqNr = y, AckNr = x+1
receive FIN+ACK and send ACK with AckNr = y+1
receive ACK

Figure 2.11: Modified three-way handshake for connection-teardown of TCP

With this mechanism each of the two directions of a full-duplex TCP connection can be separately closed.

2.3.2.3 TCP Control Variables

For each individual TCP connection, TCP uses several control variables for its built-in error correction, flow control, and congestion control. These control variables are stored in the TCP control block (TCB) of each TCP instance. The most important TCP control variables in a TCP instance are:

- **congestion window (CWND)**: The congestion window contains the current size of the sending window allowed by the congestion control. A TCP sender can send at most CWND bytes before it has to wait for acknowledgments.

- **slow start threshold (SSTHRESH)**: The current TCP congestion control (cf. Section 2.3.2.6) uses two mechanisms: slow start (SS)and congestion avoidance (CA). If the congestion window is below the slow start threshold it is increased by the maximum segment size (MSS) after an acknowledgment arrives. This is the slow-start mechanism. It is used to exponentially increase the congestion window. If the congestion window is equal to or above the slow start threshold it is increased by $\text{MSS} \cdot \text{MSS}/\text{CWND}$ bytes after the reception of an acknowledgment. This is the congestion-avoidance mechanism. It is used to linearly increase the congestion window. The value of the slow start threshold decides which of these two congestion-control mechanisms is performed to probe the currently available network capacity more or less aggressively.

- **advertised receiver window (ARWND)**: In order not to overload the TCP receiver, TCP has a built-in flow control (cf. Section 2.3.2.5). The value of this variable stores the number of bytes the TCP receiver is able to receive without being overloaded. A TCP sender is allowed to send at most ARWND bytes before it has to wait for acknowledgments of already sent but not yet acknowledged data.

- **smoothed round trip time (SRTT)**: This value contains the smoothed round trip time calculated by the last measured round trip time (RTT) and the round trip times measured in the past.

- **round trip time variance (RTTVAR)**: This value contains the round trip time variance of the TCP connection calculated over all so far send and acknowledged TCP segments.

In the following algorithmic description of TCP's error, flow, and congestion control the values $T_0, T_1, \ldots$ indicate consecutive points in time, i.e., $T_0 < T_1 < \ldots$, where events occur which have an effect on some of the control variables of a TCP sender. These events are: a TCP connection is opened or closed, a new or duplicated acknowledgment arrives at the TCP sender, or a timeout of the retransmission timer occurs in the TCP sender. Here, T_k is the time when the current event occurs, T_{k-1} is the last point in time where the control variables have been updated due to an earlier event.

2.3.2.4 TCP Error Correction

The error-correction of standard TCP is based on the ARQ mechanism Go-Back-N [22]. With Go-Back-N a TCP receiver is allowed to use cumulative acknowledgments that acknowledge the error-free reception of consecutive TCP segments. Optionally, an ARQ mechanism based on selective acknowledgments can be used that the TCP receiver has to separately acknowledge the error-free reception of every single TCP segment without the possibility of using cumulative acknowledgments. Since with this ARQ mechanism the TCP sender is able to selectively retransmit single not yet acknowledged TCP segments, this ARQ mechanism is called Selective Repeat [22]. A TCP variant following this error-correction strategy is called TCP SACK [31].

Both ARQ mechanisms require a retransmission timeout timer (RTO) for each sent segment to retransmit this TCP segment if it is not acknowledged during the lifetime of the retransmission timeout timer. In many current TCP implementations, the RTO is initialized to 6 seconds for the first sent TCP segment, the SYN segment. The RTO is updated after every reception of a TCP acknowledgment. To set the RTO to adequate values a TCP sender continuously measures the round trip time of the connection and calculates a smoothed round trip time and a round trip time variance dependent on the current measured round trip time and the round trip times measured in the past.

After the first measurement of the round trip time, the smoothed round trip time and the round trip time variance are initialized to the following values (cf. [32]):

$$\text{SRTT}(T_0) = \text{RTT}(T_0) \tag{2.1}$$

$$\text{RTTVAR}(T_0) = 1/2 \cdot \text{RTT}(T_0) \tag{2.2}$$

These values are continuously updated after every measurement of the round trip time according to the following formulas defined in [32]:

$$\text{SRTT}(T_k) = (1-\alpha) \cdot \text{SRTT}(T_{k-1}) + \alpha \cdot \text{RTT}(T_k) \tag{2.3}$$

$$\text{RTTVAR}(T_k) = (1-\beta) \cdot \text{RTTVAR}(T_{k-1}) + \beta \cdot |\text{SRTT}(T_k) - \text{RTT}(T_k)| \tag{2.4}$$

with $\alpha = 1/8$ and $\beta = 1/4$. Then the new retransmission timeout timer is set to the current smoothed round trip time plus m times the current round trip time variance according to [32]:

$$\text{RTO}(T_k) = \max\{\text{SRTT}(T_k) + m \cdot \text{RTTVAR}(T_k), 2 \cdot G\} \tag{2.5}$$

with a factor $m = 4$ and a timer granularity G which is typically set to 500 ms in many current TCP implementations. Using this value for G ensures that lost TCP segments are retransmitted without waiting too long. In addition, the probability of too early elapsed retransmission timeout timers for delayed arriving acknowledgments is very low. In future, this value for G might be decreased to 200 ms or less, since under certain network conditions lower values for the timer granularity G result in a higher performance of TCP (cf. [33]). In [34], for example, a timer granularity is proposed that should be at least one order of magnitude lower than the round trip time.

After every (consecutive) timeout of the retransmission timeout timer, the RTO is doubled. This is called the back-off mechanism of TCP's RTO calculation. It is used to adapt the RTO on a drastically increased round trip time in order to avoid spurious retransmissions if the value of the RTO is too low. For this back-off a maximum value for the RTO can be used, e.g., RTO ≤ 60 s. This maximum value can be also integrated in Equation (2.5).

2.3.2.5 TCP Flow Control

TCP's built-in flow control is window-based and works as follows. A TCP sender stores the current size of the flow-control window in the ARWND of its TCB. The ARWND is explicitly updated after every reception of a TCP acknowledgment by the value carried in the window field of the acknowledgment. This window field of a TCP acknowledgment is set by the TCP receiver to the number of bytes the TCP receiver is able or willing to receive in the near future. Since the TCP sender is allowed to send at most ARWND bytes of data before it has to wait for acknowledgments of sent TCP and not yet acknowledged segments, this flow-control mechanism ensures that the TCP receiver is protected from overload.

The basic TCP flow control is only able to handle advertised receiver windows with sizes of at most 2^{16} bytes. This might be a too low value to reach a good performance for TCP in high-speed networks. To overcome this problem, the window scale option [30] is introduced. With this option a TCP sender and its TCP receiver can negotiate a scaling factor for their advertised receiver window stored and transported in the window-field of every TCP acknowledgment. Possible ranges for this scaling factor are 1 (no scaling) to 2^{16}. Although the basic window-scale negotiation is performed only during the connection-setup phase, i.e., the window scale factor is set to a fixed value for the whole connection lifetime, it is possible and more reasonable to use another mechanism in which this TCP window scale option is sent in every segment and acknowledgment. Then, the TCP sender is able to react more accurately on highly varying free buffer space in the TCP receiver. Since the TCP window scale option requires only three additional bytes per segment or acknowledgment, respectively, the advantages that such a design choice has are large compared to the slightly increased overhead.

If the TCP window scale option is later mentioned in this dissertation, it is assumed that this second design choice is used to implement a continuous window-scale negotiation between a TCP sender and its TCP receiver. In addition, the restriction that the window-field in a SYN/ACK is never scaled (cf. [30]) is omitted, since such a limitation does not have any effects on standard TCP but can negatively influence the performance of some new NIS approaches (cf. Section 13.3).

2.3.2.6 TCP Congestion Control

TCP's built-in congestion control is window-based triggered by received or overdue TCP acknowledgments. Thus, TCP's congestion-control mechanism is combined with its error-correction mechanism. A TCP sender regards the reception of an acknowledgment that acknowledges new data as an indicator that the network is not congested. In this case, the TCP sender additively increases (AI) its congestion window to adapt to the additional available network capacity. If an acknowledgment is overdue, i.e., the retransmission timeout timer is expired or the TCP receiver has requested a missing TCP segment by sending duplicated acknowledgments (dupacks), the TCP sender assumes congestion in the network. In this case, the congestion window is multiplicatively decreased (MD) to rapidly adapt to the lower available network capacity. This AIMD scheme is used to probe the current network's capacity trying to adapt the congestion window to it in order to highly utilize but not to overload the network. In [35], it has been shown that congestion control based on such an AIMD scheme is able to stabilize the load in a network.

2.3.2.6.1 Basic TCP Congestion Control

In this section, it is explained how the basic TCP congestion control works (cf. [10]). Additional mechanisms like fast retransmit and fast recovery are explained in the following Section 2.3.2.6.2. A TCP sender stores the current value of the congestion window in the CWND of the TCB. After the connection-setup period, the CWND is initialized with the initial window (IW). A couple of years ago, the IW for the CWND was set to one MSS according to the valid standard [36] at that time. But current TCP implementations are allowed to set their initial CWND according to the following formula (cf. [37]):

$$\text{IW} = \min\left(4 \cdot \text{MSS}, \max\left(2 \cdot \text{MSS}, 4380 \text{ bytes}\right)\right) \tag{2.6}$$

Thus, for a MSS of 1460 bytes an initial CWND of 4380 bytes can be used, i.e., the TCP sender is allowed to send the first three data TCP segments of the connection after the connection establishment in a burst.

As already mentioned, the congestion-control variable SSTHRESH is used to finer control the additive increase of the CWND after the reception of an acknowledgment. In some TCP implementations, e.g., in BSD Unix, the SSTHRESH is initialized to 65535 bytes. But other values, e.g., the ARWND stored in the SYN/ACK segment are also possible. The TCP standard does not specify a precise initial value for SSTHRESH.

In the following, it is explained in detail how the AIMD scheme of the basic TCP congestion control works:

- Reception of a new TCP acknowledgment:

  If the TCP sender receives an acknowledgment that acknowledges new data, the congestion window is additively increased dependent on the current value of SSTHRESH:

$$\text{CWND}(T_k) = \text{CWND}(T_{k-1}) + \begin{cases} \text{MSS} & \text{if CWND}(T_{k-1}) < \\ & \text{SSTHRESH}(T_{k-1}) \text{ (SS)} \\ \text{MSS} \cdot \text{MSS}/\text{CWND}(T_{k-1}) & \text{if CWND}(T_{k-1}) \geq \\ & \text{SSTHRESH}(T_{k-1}) \text{ (CA)} \end{cases} \tag{2.7}$$

  The slow-start (SS) calculation allows the TCP sender to faster adapt the CWND on additional available capacity in the network by increasing the CWND by MSS bytes after every reception of an acknowledgment. The congestion-avoidance (CA) calculation is used to less aggressive adapt the CWND on additional available capacity in the network by increasing the CWND by MSS bytes after the reception of all acknowledgments during a round trip time.

  All current TCP implementations have to limit the CWND to the current ARWND to be standard-conform [37]:

$$\text{CWND}(T_k) = \min\{\text{CWND}(T_k), \text{ARWND}(T_k)\} \tag{2.8}$$

  Otherwise the TCP sender will be able to increase its CWND to values that might be much higher than the current ARWND; but then the current CWND cannot reflect the present load conditions in the network path. This will become a problem if a formerly low ARWND is suddenly noticeably increased. In this case, the TCP sender is allowed to send a lot of TCP segments—up to CWND – former ARWND TCP segments—in a single burst. This might have a negative impact on the load conditions in the network path.

- Absence of a TCP acknowledgment:

  If the TCP sender does not receive an acknowledgment during the lifetime of the retransmission timeout timer or it receives a number of acknowledgments that acknowledge already acknowledged data, i.e., duplicate acknowledgments are sent by the TCP receiver to inform the TCP sender about the reception of out-of-order TCP segments, the congestion window is multiplicatively decreased:

  $$\text{CWND}(T_k) = \text{MSS} \tag{2.9}$$

  In addition, the slow start threshold is set to the half of the currently outstanding data bytes:

  $$\text{SSTHRESH}(T_k) = \max\{\text{outstanding data at time } T_k/2, 2 \cdot \text{MSS}\} \tag{2.10}$$

It is obvious that such a congestion control based on collecting only a few network information (recently sent TCP segments have been acknowledged or not) and using them to estimate and probe the currently available network bandwidth is far from optimality regarding the throughput and the utilization of the network.

The performance of this basic TCP congestion-control scheme can be largely improved if the TCP sender is able to more adequately react on specific network events, e.g., the reception of duplicate acknowledgments, than to wait for a timeout of its retransmission timer and to decrease its congestion window to the lowest possible value. To reach this, the basic TCP congestion-control scheme is extended by introducing two new mechanisms: fast retransmit and fast recovery [10]. These mechanisms are explained in detail in the following section.

2.3.2.6.2 Advanced TCP Congestion Control: Fast Retransmit and Fast Recovery

From the TCP's sender point of view, the reception of a single duplicate acknowledgment can have several reasons, e.g.,

- one or more TCP segments are lost in the network,
- the network has reordered several TCP segments, or
- the network has replicated TCP segments and/or TCP acknowledgments.

The basic idea behind the first new mechanism, called fast retransmit, is the following: If a TCP sender receives several duplicate acknowledgments without receiving other acknowledgments in the meantime it indicates a loss of a TCP segment. In this case, the TCP sender sends the missing segment without waiting for the timeout of the retransmission timeout timer.

The basic idea behind the second new mechanism, called fast recovery, is the following: A TCP receiver is only able to send a duplicate acknowledgment if it has received a TCP segment. Thus, if a TCP sender receives a duplicate acknowledgment it can assume that a TCP segment (the one which triggered the duplicate acknowledgment) has left the network. As a consequence, the congestion in the network might be not as bad to perform slow start but to perform a new mechanism that allows to send a more adequate amount of segments.

Fast retransmit was introduced in TCP Tahoe [38] while the combination of fast retransmit and fast recovery is used in TCP Reno [10] at first and is used—sometimes modified—in many other TCP versions nowadays. The following algorithmic description shows how these new mechanisms cooperate in detail:

(1) After the reception of the third duplicate acknowledgment (the fourth identical acknowledgment), the slow start threshold is set to the value given by Equation (2.10).

(2) The lost segment is retransmitted and the congestion window is set to

$$\text{CWND}(T_k) = \text{SSTHRESH}(T_k) + 3 \cdot \text{MSS} \tag{2.11}$$

using the new calculated slow start threshold from step (1) and taking into account the TCP segments which have left the network and have caused the TCP receiver to send duplicate acknowledgments.

(3) For each additionally received duplicate acknowledgment the TCP sender increases its congestion window by one MSS. This step artificially inflates the congestion window to reflect TCP segments which have left the network.

(4) A new TCP segment is transmitted if it is allowed by the current sending window, the minimum of the current congestion window and the current advertised receiver window.

(5) If the TCP sender receives an acknowledgment that acknowledges new data the congestion window is set to the slow start threshold from step (1), i.e.,

$$\text{CWND}(T_{k+l}) = \text{SSTHRESH}(T_k). \tag{2.12}$$

with $l > 0$. This deflates the congestion window. Usually, the new acknowledgment acknowledges the retransmitted TCP segment from step (2) one round trip time after the fast retransmit / fast recovery procedure has been initiated.

This algorithm does not recover very efficiently from multiple segment losses during a single flight of segments. To handle these events more accurately, an improved algorithm is developed which is explained in the following section.

2.3.2.6.3 TCP NewReno

Although TCP is used for decades, its development is not finished so far. In recent years, several variants of TCP have been proposed and some of them have been implemented in the operating systems running in Internet end systems. Therefore, various TCP variants and different TCP implementations are simultaneously used in today's Internet environment.

In this dissertation, the TCP-variant TCP NewReno [11] is used to investigate the influence of several network-information sharing approaches on the performance of TCP. TCP NewReno is one of the most progressive TCP variants and it is widely used in the real Internet, e.g., the TCP implementation of Linux is based on a modificated TCP NewReno (cf. Section 10.2). TCP NewReno performs Go-Back-N as the error-correction strategy. In addition, TCP NewReno uses the basic TCP congestion-control algorithms in combination with a modified fast retransmit / fast recovery mechanism which is explained in detail in the following algorithmic description:

(1) If the TCP sender is not already in the fast recovery procedure and the third duplicate acknowledgment arrives, the slow start threshold is set to the value given by Equation (2.10). In addition, the variable "recover" is set to the highest sequence number sent so far.

(2) The lost segment is retransmitted and the congestion window is set to

$$\text{CWND}(T_k) = \text{SSTHRESH}(T_k) + 3 \cdot \text{MSS} \tag{2.13}$$

using the new calculated slow start threshold from step (1) and taking into account the TCP segments which have left the network and have caused the TCP receiver to send duplicate acknowledgments.

(3) For each additionally received duplicate acknowledgment the TCP sender increases its congestion window by one MSS. This step artificially inflates the congestion window to reflect TCP segments which have left the network.

(4) A new TCP segment is transmitted if it is allowed by the current sending window, the minimum of the current congestion window and the current advertised receiver window.

(5) If the TCP sender receives an acknowledgment that acknowledges new data the following steps are performed dependent on the type of the received acknowledgment:

   (a) If the acknowledgment does not acknowledge all data up to and including the sequence number "recover", a partial acknowledgment has been received. In this case, the next unacknowledged segment is retransmitted and the congestion window is decreased by the amount of the new acknowledged data and increased by one MSS, i.e.,

$$\text{CWND}(T_{k+i}) = \text{CWND}(T_{k+i-1}) - \text{new acknowledged data at time } T_{k+i} + \text{MSS} \tag{2.14}$$

   for all $i < l$. In addition, a new TCP segment is sent if it is allowed by the new congestion window. The fast recovery procedure is continued. And the first partial acknowledgment received during the fast recovery procedure is used to reset the retransmission timeout timer.

   (b) If the acknowledgment acknowledges all outstanding data up to and including the sequence number "recover", the fast recovery procedure is finished. In this case, the congestion window is set to one of the following values:

$$\text{CWND}(T_{k+l}) = \begin{cases} \min\{\text{SSTHRESH}(T_k), \\ \qquad \text{outstanding data at time } T_{k+l} + \text{MSS}\} & \text{version I} \\ \text{SSTHRESH}(T_k) & \text{version II} \end{cases} \tag{2.15}$$

In both versions, SSTHRESH is the slow start threshold set in step (1). And in version I, the amount of outstanding data is set to the current value. One problem can occur with the second version if the amount of outstanding data is much lower than the new congestion window. Then, the TCP version should perform a pacing algorithm in order to avoid that the TCP sender sends a burst of new data.

In both cases, the congestion window is (partially) deflated.

As an example, Figure 2.12 shows the process of the congestion window and the current sequence number of a TCP NewReno connection.

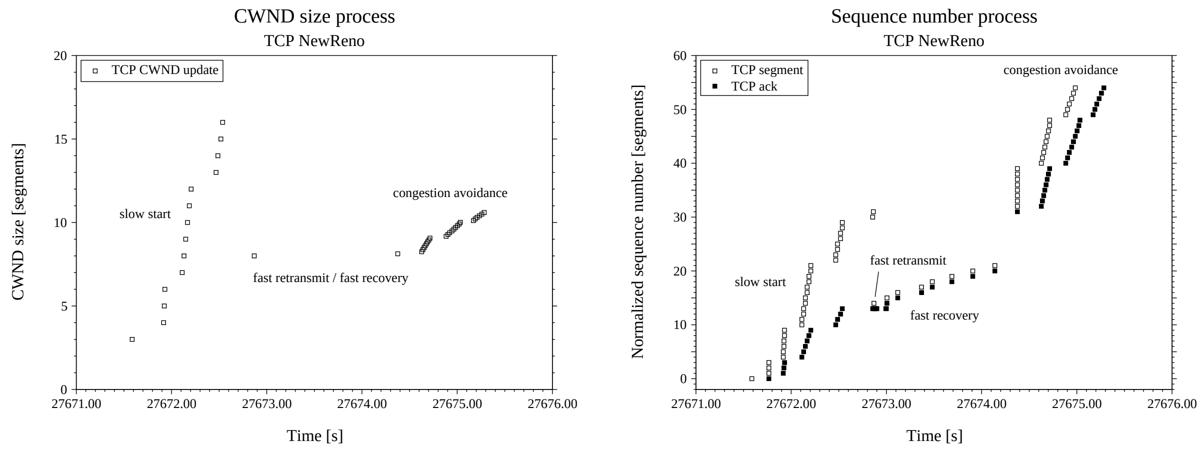


Figure 2.12: TCP NewReno congestion control of a TCP connection

In this figure, slow start, fast retransmit, fast recovery, and congestion avoidance can be easily identified. One very interesting event in this example-TCP-NewReno connection is the reception of a cumulative acknowledgment at time 27674.38 s. This specific cumulative acknowledgment acknowledges all so far outstanding TCP segments at once with the result that the TCP sender is allowed to send 8 maximum-sized new TCP segments in a burst.

2.3.2.6.4 TCP Congestion-Control Extensions

The pure TCP end-to-end congestion-control scheme can be assisted by some rudimentary implicit or explicit congestion-control feedback from routers or by introducing more complex proxy-based approaches. Regarding the feedback mechanism, two standardized approaches Random Early Detection (RED) [39, 14] and Explicit Congestion Notification (ECN) [15] and the new proposal AntiECN (AECN) [40] are briefly explained in the following. Afterwards, the main ideas behind proxy-based approaches for TCP [41] are described.

Random Early Detection (RED) The simplest queue type IP routers can be equipped with and are equipped with in the current Internet are droptail queues [33]. With these queues an arriving packet is only discarded if the queue is full. Otherwise, the packet is temporarily stored in the queue and later forwarded to its destination. This queue strategy tends to discard many (consecutive) packets in case of congestion. Thus, a lot of TCP connections are affected by packet losses nearly at the same time that a higher correlation of packet-loss events for these TCP connections can be observed. In general, this packet-loss correlation has a negative influence on the fairness and throughput of TCP connections traversing such a router queue [14]. In addition, like every other passive queue management droptail queues tend to have larger mean queue lengths compared to more sophisticated queueing strategies, e.g., queues with an active queue management, since implicit congestion feedback (packet discards) for TCP senders to enable them to reduce their sending window is only performed if the router queue is full that packets get lost.

These negative effects of droptail queues on the performance of TCP connections can be reduced if an active queue management like RED [39, 14] is used in the router queues. The basic RED algorithm measures the mean queue length as the control variable and compares it with two threshold queue length parameters. If the mean queue length is below the lower queue threshold, all arriving packets are temporarily stored in the router queue and are later forwarded. If the mean queue length is above the upper queue threshold, all arriving packets are discarded. If the mean queue length is between the lower and the upper threshold, arriving packets are independently discarded with a probability that increases linearly from zero to $p_{\max}$, known as drop function. As a result in case of impending congestion in a RED queue, only some randomly selected TCP connections traversing the router queue are affected by packet losses and reduce their sending window according to the standard TCP congestion control. Thus, with RED the mean queue length and delay are decreased and the fairness of TCP connections traversing the router queue is increased [14]. There exist many RED variants with different calculations of the control variable and/or the drop function or the RED parameters. In [33], the most important RED variants are described and compared.

Explicit Congestion Notification (ECN) If a router is equipped with ECN [15], it marks packets and forwards them instead of discarding them in case of (impending) congestion due to a highly loaded queue. Packets are marked by an ECN-capable router by setting a single bit in the type-of-service field in its IP header, the explicit congestion (EC) flag. If a TCP receiver receives such a marked IP packet it acknowledges the TCP segment (the IP payload) with a marked TCP acknowledgment. This is a standard TCP acknowledgment in which the last bit of the reserved bits is set to one, i.e., this bit is used as an explicit-congestion echo (ECE) flag. If a TCP sender receives such a marked TCP acknowledgment it performs its congestion control algorithm just as the acknowledged TCP segment has been discarded by the router. Thus, the sending window of the TCP sender can be faster reduced without packet losses.

With both standardized mechanisms the TCP senders are able to react on (impending) congestion in the network by decreasing their sending windows in order to prevent or reduce congestion in the network. The ECN mechanism has the advantage that no IP packets have to be discarded to inform the TCP senders about the impending congestion. But the disadvantage of the ECN mechanism is that it requires minor changes in the IP as well as in the TCP protocol layer of the end systems. In contrast to that the RED approach is transparent for the end systems.

In recent years, a lot of router manufacturers have equipped their routers with RED and/or ECN. But currently, most network operators do not use these router features in practice (cf. [42]).

AntiECN (AECN) AntiECN [40] is a new proposal which is related to ECN. But in contrast to ECN where ECN-capable routers are able to explicitly decrease the sending window of a TCP sender at any time, the AECN-capable routers are able to explicitly allow a TCP sender to additively increase its sending window by a fraction of its current congestion window during the congestion-avoidance phase.

In order to prevent the network from congestion, the AECN mechanism requires that all routers along the network path of a TCP connection are AECN-capable. This is checked during the setup phase of a TCP connection. If all routers in the network path of a TCP connection are AECN-capable, the AECN mechanism can be used. Otherwise, the AECN mechanism cannot be used and standard TCP congestion control has to be performed.

The AECN mechanism works as follows. The TCP header of each TCP segment contains an AECN bit which is initially set to one by a TCP sender and can be changed by any AECN-capable router in the network path. To reach this, each router along the network path continuously measures its current load. If the current load is beyond a given threshold and the router wants to allow a TCP sender to increase its congestion window according to AECN, the router does not change the AECN bit in the header of a TCP segment. Otherwise, the AECN bit is set to zero. The AECN bit in a TCP segment arrived at a TCP receiver is copied and sent back to the TCP sender in a TCP acknowledgment. If a TCP sender receives an acknowledgment with a set AECN bit, it additively increases its congestion window by a fraction of its current congestion window. Otherwise, the TCP sender performs the normal congestion control during the congestion-avoidance phase. AECN reaches the same performance than standard TCP over moderately to highly loaded network paths but has huge throughput gains over underutilized (high bandwidth-delay) network paths (cf. [40]).

Proxy-based TCP extensions Another possibility to extend standard TCP congestion control is to use proxy-based approaches [41]. This is done by splitting a single TCP connection at a network node called proxy in two parts: the first part ranges from the original sender to the proxy and the second part ranges from the proxy to the original receiver. Inside the proxy for each splitted TCP connection a virtual receiver for the first part of the connection and a virtual sender for the second part of the connection is introduced.

The general advantage such a proxy-based approach can have is that the two parts of the original TCP connection can be separately controlled with a (much) shorter response time on triggering error- or congestion-control events that a higher performance of the proxy solution compared to the original end-to-end communication can be expected. Therefore, these proxies are called performance-enhancing proxies (PEPs) to distinguish them from other proxy types, e.g., mobility-enabling proxies (MEPs) used, for example, as home and foreign agents in Mobile IP [43].

PEPs are in general beneficial if they are located in specific network nodes that separate network parts with (very) different network characteristics, e.g., in gateways between wired and wireless regions of a network [33]. Examples for such PEP approaches are approaches related to Indirect-TCP (I-TCP) [44], Split-TCP [45], and—as a very special variant of them—the Remote Socket Architecture (ReSoA) [46] approach developed and evaluated at the TKN institute. The two main drawbacks of these PEP approaches are their comparatively large additional complexity that is needed inside the proxy to handle the state of the virtual senders and receivers and that they break the end-to-end semantic of TCP (TCP segments can be acknowledged by the virtual receiver at the proxy before they have arrived at the original receiver).

2.3.2.6.5 Experimental TCP Congestion Control

TCP Vegas [47] and TCP Westwood [48] are two experimental TCP variants that try to directly adapt the CWND and SSTHRESH on the current expected throughput or bandwidth, respectively, of the network path. TCP Vegas calculates the current throughput on the basis of the current CWND and the current RTT and compares it with the expected throughput (the pipe capacity), calculated on the basis of the current CWND and the lowest so far measured RTT. If the difference between the expected and the current throughput is below a threshold, the CWND is linearly increased. And if the difference between the expected and the current throughput is above another threshold, the CWND is linearly decreased. Otherwise, the CWND is not changed. TCP Westwood estimates the current bandwidth in the network path by counting the arrival and amount of acknowledged data of acknowledgments during a measurement interval. After a packet loss has been indicated, the CWND and SSTHRESH are set based on the estimated bandwidth.

In addition, several experimental TCP variants have been developed that allow a better utilization of high-speed network paths by TCP connections following adapted or completely newly developed congestion-control algorithms. Examples are Scalable TCP [49], HighSpeed TCP [50], and FAST TCP [51]. Scalable TCP and HighSpeed TCP adapt only the two parameters of standard TCP NewReno that are used to additively increase and multiplicatively decrease the current congestion window dependent on the different congestion-control events during the congestion-avoidance phase. But FAST TCP uses a completely new calculation of the congestion window based on the estimation of the current mean and minimum round trip time, the current mean queueing delay, and the current mean packet loss rate in the network path.

2.3.2.7 TCP Performance Metrics

This section contains a formal description of the main performance metrics used to evaluate TCP. These performance metrics are the throughput of a single TCP connection, the throughput of a set of TCP connections, and the fairness between a set of TCP connections.

2.3.2.7.1 Throughput

The throughput *TP* of a single TCP connection, measured in units of segments per second, in a given time interval d is defined as the number of segments s sent and acknowledged during this time interval:

$$\mathit{TP} = \frac{s}{d} \qquad [\text{segments}/\text{second}] \tag{2.16}$$

For a set of n TCP connections two different mean throughput metrics can be used whose calculations are explained in the following:

- Computation of the overall mean throughput $\overline{\mathit{TPO}}$: The sum of sent and acknowledged segments of n TCP connections is divided by the overall duration of these TCP connections:

$$\overline{\mathit{TPO}} = \frac{\sum\limits_{i=1}^{n} s_i}{\sum\limits_{i=1}^{n} d_i} \qquad \text{[segments/second]} \tag{2.17}$$

- Computation of the connection-oriented mean throughput $\overline{\mathit{TPC}}$: For each of the n TCP connections a mean throughput *TP* is calculated. All these mean throughput values are then used to compute the connection-oriented mean throughput of the n TCP connections by a normal non-weighted arithmetic mean calculation independent of the number of segments sent and acknowledged by each of the TCP connections:

$$\overline{\mathit{TPC}} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{s_i}{d_i} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathit{TP}_i \qquad \text{[segments/second]} \tag{2.18}$$

The first mean throughput metric is the more important one, since with this mean throughput metric the overall mean throughput of the set of TCP connections can be evaluated. The second mean throughput metric gives a connection-oriented mean throughput; it can be understood as the mean throughput a single TCP connection of the set can expect.

2.3.2.7.2 Fairness

The fairness between a set of n TCP connections is defined as follows: If n TCP connections are established in parallel for a period of time and reach the throughputs TP_i, $1 \leq i \leq n$, in this period of time, then for these TCP connections a fairness index can be computed [1]:

$$I_f = \frac{\left(\sum\limits_{i=1}^{n} \mathit{TP}_i\right)^2}{n \cdot \sum\limits_{i=1}^{n} \mathit{TP}_i^2} \quad \text{with} \quad \frac{1}{n}\,\text{(worst case)} \leq I_f \leq 1\,\text{(optimal case)}. \tag{2.19}$$

A fairness index of $1/n$ denotes that one of the n concurrent TCP connection of the set gets the entire available bandwidth, i.e., it is the only TCP connection with a throughput larger than zero. A fairness index of 1 means that all n concurrent TCP connections of the set get the same portion of the available bandwidth, i.e., they reach all the same throughput larger than zero.

The mean fairness over n times of concurrencies is computed by a normal non-weighted arithmetic mean calculation independent of the duration of the concurrency periods:

$$\overline{I_f} = \frac{1}{n} \cdot \sum_{i=1}^{n} I_{fi} \tag{2.20}$$

Throughout the rest of this dissertation, the unit for the control variables CWND and SSTHRESH is segments with a maximum segment size and not bytes.

2.3.2.8 Mathematical Models of a Single TCP Connection

In [52], a simple mathematical model of a single TCP connection is developed that is based on period patterns regarding the packet losses and the dynamics of the congestion window. From this periodic model a formula, known as the *inverse square-root* p law, has been derived that calculates the throughput a single TCP connection can expect dependent on the round trip time and the packet loss rate p in the network path:

$$\overline{TP}(p) = \frac{1}{\text{RTT}} \cdot \sqrt{\frac{3}{2p}} \tag{2.21}$$

In [53], a more complex mathematical model of a single TCP connection is developed that considers packet loss events at the TCP sender triggered by dupacks and timeouts. This detailed packet-loss model is used to derive an approximation of the mean throughput a single TCP connection can expect dependent on the advertised receiver window ARWND, the round trip time RTT, the timeout period TOP, and the mean packet loss rate p in the network path:

$$\overline{TP}(p) \approx \min\left(\frac{\text{ARWND}}{\text{RTT}}, \frac{1}{\text{RTT} \cdot \sqrt{\frac{2p}{3}} + \text{TOP} \cdot \min\left(1, 3\sqrt{\frac{3p}{8}}\right) \cdot p \cdot (1 + 32p^2)}\right) \tag{2.22}$$

This formula is also only valid for long-term TCP connections. If the packet loss rate p in the network path is very low, the second term in the minimum-calculation of Equation (2.22) is an approximation of the inverse square-root p law.

From both formulas it can be de derived that even if two TCP connections traverse the same bottleneck link in the network, their expected mean throughput can largely vary dependent on their potentially different round trip times.

2.3.3 User Datagram Protocol (UDP)

Since the investigations of different NIS approaches in this dissertation are not considering UDP or TCP-friendly protocols based on UDP, the description of UDP is rather compact and no TCP-friendly approaches are described.

UDP is a connection-less transport protocol. That means that payload can be immediately transferred between UDP instances without the necessity to set up a connection as it is required, for example, if TCP is used. In contrast to TCP, UDP has no built-in error correction functionalities. Thus, if a reliable data transfer is required for the data transfer over UDP, error correction is left to the application. In addition, UDP has no built-in flow and congestion control. But there exist some mechanisms that provide a congestion control for UDP flows to prevent that UDP flows overload the network, e.g., TCP-Friendly Rate Control (TFRC) [54]. Since these mechanisms are developed that UDP flows are as aggressive to the network as TCP streams, they are called TCP-friendly. Several of such TCP-friendly approaches have been developed by using the TCP throughput-approximation given in Equation 2.22.

2.3.3.1 UDP PDU

The PDU of UDP, called UDP datagram, is shown in Figure 2.13. It consists of a header and the payload

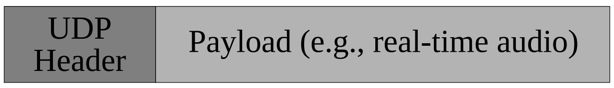


Figure 2.13: UDP datagram

carrying, for example, real-time audio data. The UDP header is shown in Figure 2.14. In the following,

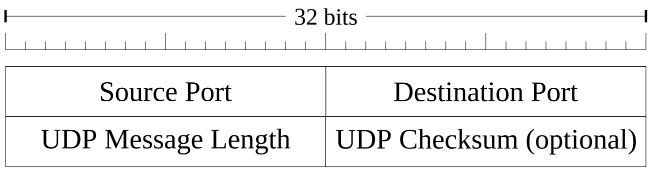


Figure 2.14: UDP header

the functionalities and possible values of each field in the UDP header of a UDP datagram are described: The first two fields in the UDP header contain the source and destination port of the UDP flow. Together with the source and destination address of the end systems stored in the IP header these two values are used to clearly identify the two processes in the end systems which exchange data using the UDP flow. The next field stores the overall length of the UDP message, i.e., the sum of the header and the payload length of the UDP datagram. And the last field of the UDP header contains the checksum of the UDP datagram. The use of this checksum is optional if UDP is working on top of IP but it is mandatory if IPv6 is the underlying network protocol.

Chapter 3

Network-Information Sharing in the Internet

3.1 Introduction

In the last chapter, the standardized TCP congestion-control mechanisms used in the current Internet have been described in detail. In addition, it has been explained how the TCP end-to-end congestion-control scheme can be enhanced by using rudimentary standardized congestion-control feedback mechanisms from routers like RED or ECN. Even in today's Internet, the performance of such an enhanced congestion control might be far from optimality due to the insufficient or outdated information about the current network conditions collected in the TCP instances running in the end systems. This problem will be strengthen in future IP-based networks where the transparent integration of different wired and wireless access technologies with their specific bandwidth, delay, and error characteristics will play an important role. Therefore, it might be advantageous in terms of improving the overall performance of the Internet to assist TCP's end-to-end congestion control with more appropriate mechanisms based on network-information sharing (NIS). This chapter describes in general the functionalities and properties of such NIS approaches for TCP in the current Internet and future IP-based networks. In addition, a brief overview about the metrics and simulation models used in this dissertation for the performance evaluation of some NIS approaches is given.

3.2 Fundamental Types of Network-Information Sharing for TCP

Two fundamental types of NIS approaches can be used to assist TCP's congestion control:

(1) network information can be shared between senders of TCP connections in an Internet end system,

(2) network information can be shared between Internet routers and end systems.

In general, approaches of both NIS types can be combined to develop a hybrid NIS approach in the Internet. But such hybrid NIS approaches are not in the main focus of this dissertation.

In the next two Sections 3.3 and 3.4, these two NIS types are generally considered with regard to their properties, functionalities, complexity, applicability in the current Internet environment, and potential for increasing the performance of standard TCP. In Section 3.5, a summary of these considerations is given.

3.3 Network-Information Sharing in an Internet End System

3.3.1 Introduction

All TCP senders of TCP connections in a single end system act separately and independently from each other. This means, for example, that these TCP senders do not share their collected information about the current network conditions. As a result, the observed performance of some or all of the TCP connections can be far from optimality, since the collected network information in some or all the TCP senders might not accurately map the current load conditions in the Internet. Therefore, to reach a better performance it might be promising to share network information between some TCP senders of an end system (cf. Figure 3.1).

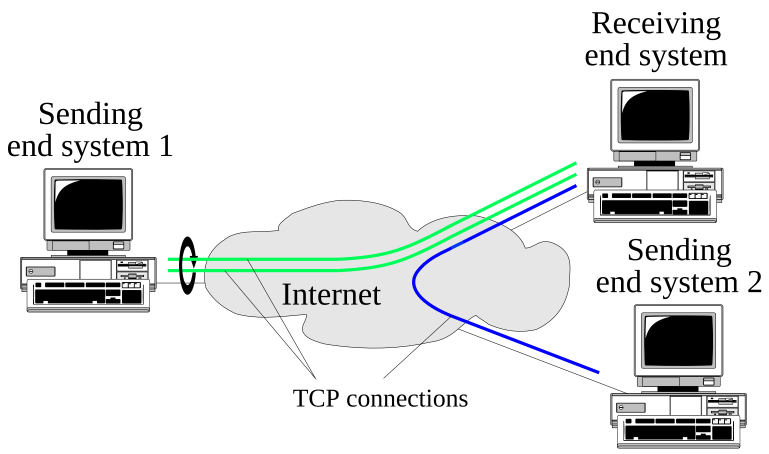


Figure 3.1: Network-information sharing between TCP connections in an Internet end system

3.3.2 Ensemble of TCP Connections

Such a network-information sharing is only reasonable between TCP senders of an end system which have the same TCP receiver or TCP receivers in the same part of the network, since then these TCP connections traverse the same path and bottleneck router or link, respectively, in the Internet. Then, these TCP connections can form an ensemble of TCP connections (or an ensemble for short) and share network information. Measurements [55] in a real WWW server have shown that it is not unlikely to have such ensembles of TCP connections, since several browser implementations often use several TCP connections at the same time to download the objects of a single WWW page in parallel. In addition, in [56] other measurements are considered by observing the IP addresses of IP packets traversing real links which generally see a lot of traffic aggregation, e.g., access links to universities, to popular web servers, and to ISPs. The method behind these measurements is to trace the IP addresses of IP packets traversing a single link and group them into address-ensembles with identical p-prefixes of their IP addresses, with $0 \leq p \leq 32$ (IPv4 is used). The main results are that each link has its own (specific) pattern of p-prefix address-ensembles, but the probability of these p-prefix address-ensembles follows a distribution that can be expressed as a multifractal model known as Cantor's Dust. In other words, it is not unlikely to observe IP addresses in each of these links that can be grouped into address-ensembles with the same p-prefix for $p = 8$, 16, or 20 (with a heavily decreasing probability for higher p's). And these observations even show a short-term stability.

3.3.3 Types of Network-Information Sharing Approaches in an Internet End System

Network-information sharing can be done once at the start of every new TCP connection entering an ensemble or—as an extension—continuously during the whole lifetime of TCP connections in an ensemble. The former approach is called by me one-time network-information sharing while the latter approach is called by me continuous network-information sharing or common congestion control, respectively. Which, how, and when network information between TCP connections of an ensemble is shared depends on the controller of the network-information sharing or common congestion control approach applied in the end system. Such a controller is an abstract entity which needs to be specified further to determine a concrete, implementable, and testable functionality.

3.3.4 Common Congestion Control

In this section, the main tasks and design principles and aspects of a common congestion controller are described. Topics related to a pure one-time network-information sharing controller are also included in this description. The focus is on a common congestion controller located in the transport layer of the protocol stack of an end system.

One-time network-information sharing or common congestion-control approaches can be also established in the application layer of the Internet protocol stack. For example, a well-known specific type of network-information sharing approach located in the application layer of the Internet protocol stack is the Hypertext Transfer Protocol (HTTP) in its current version 1.1 [17, 18]. It is used in many WWW servers and provides a network-information sharing mechanism for multiple HTTP transactions to one destination. But compared to this, a common congestion control located in the transport layer features much more functionalities. It is able to share network information not only between multiple HTTP transactions to one destination but also between multiple TCP-based transmissions to one destination or to many destinations in the same subnetwork. In addition, a common congestion controller located in the transport layer can be implemented to allow a transparent integration of its features for all applications running on the end system. These are the reasons why in this dissertation only one-time network-information sharing and common congestion-control approaches are considered in more detail that are located in the transport layer of the protocol stack in Internet end systems.

In [57], a general description of one-time network-information sharing approaches for TCP is given. But general implications of common congestion-control approaches for TCP are also discussed. Both approaches are informally described in the context of the TCP Control Block Interdependence (TCBI) where TCP senders share network information stored in the control variables of their TCBs. These TCP senders can be located in a single end system or distributed over several end systems in a local region, e.g., a LAN. Thus, the TCBI approach can be used to share network information between TCP senders on a host-to-host, a host-to-LAN, a LAN-to-host, and on a LAN-to-LAN basis. End systems that are equipped with TCBI-capabilities are called TCBI-capable end systems (or TCBI end systems for short). A TCBI controller in each of these end systems manages the network-information sharing. The TCBI approach can be considered as the basis for all NIS approaches which support a similar kind of network-information sharing between TCP connections in a transparent way.

3.3.4.1 Main Tasks

A common congestion controller's job can be divided in three main tasks:

(1) **Decision which TCP connections can form an ensemble**: A common congestion controller has to decide which TCP connections can form an ensemble to enable their senders to share network information to be jointly controlled.

(2) **One-time network-information sharing**: A common congestion controller has to manage the one-time network-information sharing between *existing* or *recently closed* TCP connections of an ensemble and a *new* TCP connection entering this particular ensemble. In the context of TCBI [57], the first possibility of one-time network-information sharing is called ensemble-TCBI while the second one is called temporal-TCBI.

(3) **Continuous network-information sharing**: A common congestion controller is responsible for the continuous network-information sharing between *existing* and *concurrently operating* TCP connections of an ensemble to reach a common congestion control between all TCP connections of the ensemble.

Only the first two tasks are performed by one-time network-information sharing controllers.

3.3.4.2 Design Principles and Aspects

This section considers in detail the main design principles and aspects of a common congestion controller in an Internet end system.

3.3.4.2.1 Ensemble-Decision Criterion

One of the most important design principles of a common congestion controller is the criterion that is used to decide which TCP connections can form an ensemble. In the simplest case, this decision is made based on the IP addresses of the receiving end systems. A necessary assumption for this approach is that the IP addresses of the receiving end systems are related to the locations of the receiving end systems in the Internet. But this cannot always be ensured. For example, some standardized IP-based mechanisms, e.g., Network Address Translation (NAT) [58] or Mobile IP [54], violate the address-location relation of IP addresses with the result that IP addresses of a single subnetwork can belong to receiving end systems with very different locations. Then, this simple decision-criterion for forming an ensemble cannot be used any longer. But even if the IP addresses of the receiving end systems reflect the location of these end systems, it is not easy to find out which TCP connections can form an ensemble. If the precise segmentation of a class A, B, or C network (cf. Section 2.2.3) in subnetworks is not known, only the network part of a class A, B, or C IP address can be used for that. But this might be too inaccurate, since, for example, a whole class B network usually consists of many LANs with possibly different (currently) available bandwidths. In general, using a NIS approach between TCP connections with TCP receivers located in end systems of such a large network part cannot accurately work.

Since the IP addresses of the receiving end systems cannot always be used as a decision-criterion for composing an ensemble of TCP connections, other criteria have to be considered. One new decision-criterion is based on measurements of the current network conditions observed by the TCP senders and works as follows: If measurements show that some concurrent TCP connections of an end system experience (nearly) the same packet loss rate and/or round trip time delay during a measurement period, i.e., the packet loss patterns of these TCP connections are highly positively correlated, it can be assumed that these TCP connections traverse the same bottleneck router in the network. As a result, these TCP connections can form an ensemble and can be jointly controlled [59], otherwise the TCP connections cannot form an ensemble (anymore). These measurements can be done in-band, i.e., during the lifetime of the TCP connections, or out-band by sending probing packets to the receivers which send them back. These probing packets can be sent either to all receivers before the TCP connections start to send their payloads or continuously to some chosen receivers. The former probing approach requires an additional time for its decision where the latter probing approach can only work if the chosen receivers are carefully selected. A combination of both probing approaches, i.e., the latter approach combined with the former approach as a fallback mechanism, might be a promising solution. Another idea is to observe the feedback the TCP senders of a single end system receive from routers that support NIS approaches of the second type. If the feedback from the routers is similar for some TCP connections over a sufficient large time, their senders can be jointly controlled. This approach has the advantage that the decision whether TCP connections belong to an ensemble or not can be much faster answered than with the above explained approach based on packet loss measurements.

3.3.4.2.2 Creation of an Ensemble

If it has been found out that receiving end systems are located in the same subnetwork that TCP connections from one sending end system to these receiving end systems can (statically) form an ensemble, it is not always desirable that these TCP connections should form an ensemble. A counter-example is a subnetwork consisting of a radio access network (RAN) using a wireless access technology. In general, wireless channels in a RAN have the inherent properties that their transmission quality is dependent on the location of the communicating (mobile) end systems and (largely) varies over the time. Thus, each TCP sender of the TCP connections can observe (very) different current network conditions in (the wireless hop of) its network path. It is obvious for performance reasons that those TCP connections should not form an ensemble. Thus, in order to avoid inadequately (jointly) controlled TCP connections, a common congestion controller has to deal also with these effects of a wireless environment. This means, for example, that it is not always sufficient to form an ensemble by using only the localization information of the receiving end systems, since these end systems—although they are located in the same RAN—can have very different (current) transmission qualities in their wireless channels. Therefore, a common congestion controller should permanently observe the network conditions collected by the TCP senders and should dynamically decide whether TCP connections to different receivers in the same RAN can form an ensemble and are jointly controlled or not.

3.3.4.2.3 Interactions with Applications

A common congestion controller can be transparent or non-transparent for the applications running on the end system. With a non-transparent common congestion control the applications can be informed by the transport layer about the current network conditions that they can adapt their sending rate according to these network information. But this requires a new application programming interface (API) between the application layer and the transport layer, i.e., the applications have to be revised.

3.3.4.2.4 Mechanisms to Share Network Information

A common congestion controller reuses network information discovered from senders of TCP connections of an ensemble for a sender of a new TCP connection entering this ensemble. These network information can be obtained from senders of currently existing, concurrent TCP connections or from senders of recently closed TCP connections. In the second case, the network information is cached in the transport layer and temporarily shared. Due to the generally dynamically changing network conditions in the Internet, temporarily shared network information are only valid for a shorter period of time. Measurements [60] have shown that with a proper observation method for the available bandwidth on a path seen by one TCP connection the value for the observed bandwidth can be used as a fairly good prediction for the future available bandwidth for other TCP connections on the same path for a period of time in the order of minutes. But this interesting result might not hold in general and in future. Thus, using of cached network information must be combined with a maximum lifetime for these network information. During this lifetime the cached network information should be “aged” to soften their influence on the aggressiveness of a TCP sender using this cached network information. How this can be accurately done is an open problem.

3.3.4.2.5 Control Strategies and Control Variables

There exist many strategies how TCP senders of an ensemble can be jointly controlled. For example, one common congestion controller shares only control variables between senders of the TCP connections of an ensemble; the TCP connections of an ensemble are rather slightly coupled. Another common congestion controller uses an additional single packet queue combined with a packet scheduler for all packets generated by the TCP senders of an ensemble. Then, the TCP connections of an ensemble are stronger coupled.

In this context, it is important to consider which control variables are shared by a common congestion controller among the senders of TCP connections of an ensemble. For example, one common congestion controller shares only congestion-control variables, e.g., CWND and SSTHRESH, while another common congestion controller shares also error-control variables, e.g., SRTT and RTTVAR, among the senders of TCP connections of an ensemble. Therefore, the choice of the control algorithms, the control structures, and the type of jointly controlled variables are important design principles for a common congestion controller.

3.3.4.2.6 Complexity

Another important design aspect is the computational complexity of a common congestion controller, i.e., its additional time and space consumption compared to the standard TCP controller used in current Internet end systems.

The following Table 3.1 summarizes the design principles and aspects of one-time network-information sharing and common congestion-control approaches in an Internet end system.

Table 3.1: Design principles and aspects of a network-information sharing controller in an Internet end system

Design principle / aspect	Possibilities
Ensemble-decision criterion	IP addresses, measurements, network-assisted
Ensemble-creation	static / dynamic
Network-information sharing	one-time (temporal, ensemble) / continuous
Interactions with applications	transparent / non-transparent
Control strategies	share control variables and/or control entities
Control variables	congestion-control and/or error-control variables
Complexity	time and space in end systems (or other network nodes)

3.3.5 Performance Considerations

In the following, it is described when a common congestion controller is able to improve the performance of TCP connections:

- A common congestion controller is used to shorten the transient start phase of every TCP connection with potentially smaller congestion windows due to the TCP slow-start algorithm. Particularly for short TCP connections with only a few segments to send, e.g., TCP connections used for transactions or used for the transfer of short WWW pages, the common congestion controller might be very useful, since for each of these TCP connections the transient start phase is a large part of its whole lifetime which has a high and negative influence on the overall throughput of such a TCP connection.

- Although a common congestion controller might be very useful for short TCP connections, an accurate collection of adequate information about the network in an end system is only possible if the end system has many TCP connections in parallel or one after another at short time intervals and at least some of them should be longer TCP connections with dozens of segments to be transmitted. Because only those longer TCP connections have a chance at all to increase their congestion window to a value that represents the currently available bandwidth in the network path to remote receiving end system(s) so that shorter TCP connections can benefit from it if they are jointly controlled.
- Another interesting aspect is that only those TCP connections can benefit from a common congestion controller which have a sufficient long round trip time between the sending end system and the remote receiving end system(s) and therefore can make use of a higher (initial) congestion window. Other TCP connections with a short round trip time, e.g., intra-LAN TCP connections, cannot take advantage of a higher (initial) congestion window, since in this case the sending end system is only able to send only a few TCP segments during the short round trip time.

In general, a typical Internet end system rarely has as many TCP connections that a common congestion-control approach can be used by expecting a noticeable increase of the mean throughput of its TCP connections. But, for example, a highly frequented WWW server or a proxy server have a much higher probability to improve the fairness and the mean throughput of some of their TCP connections if they are equipped with common congestion-control capabilities.

3.3.6 Potential of Network-Information Sharing in an Internet End System

The following Figure 3.2 shows the potential an approach based on (one-time) network-information sharing in an end system can have in improving the performance of a single TCP connection. The performance metric used for the comparison of standard TCP with the NIS approach is the overall throughput of the connection measured in segments sent and acknowledged during the connection lifetime with the connection-setup phase included. In this example-scenario a single TCP connection is created to transfer 10 segments from a sender to a remote receiver over a network path with a minimum round trip time of 100 ms. After the TCP connection has been established, the standard TCP controller performs slow start and reaches an overall throughput of 25 segments per second. But with an approach based on (one-time) network-information sharing the TCP sender is allowed—in the best case—to send all 10 segments at once. Since such a burst of segments would cause some negative effects on the dynamics of the queue lengths in the routers, an additional pacing algorithm is used that temporarily disperses the segments of a single burst. Then, this NIS-assisted TCP sender reaches a throughput of 41.24 segments per second and outperforms standard TCP by approximately 65 %.

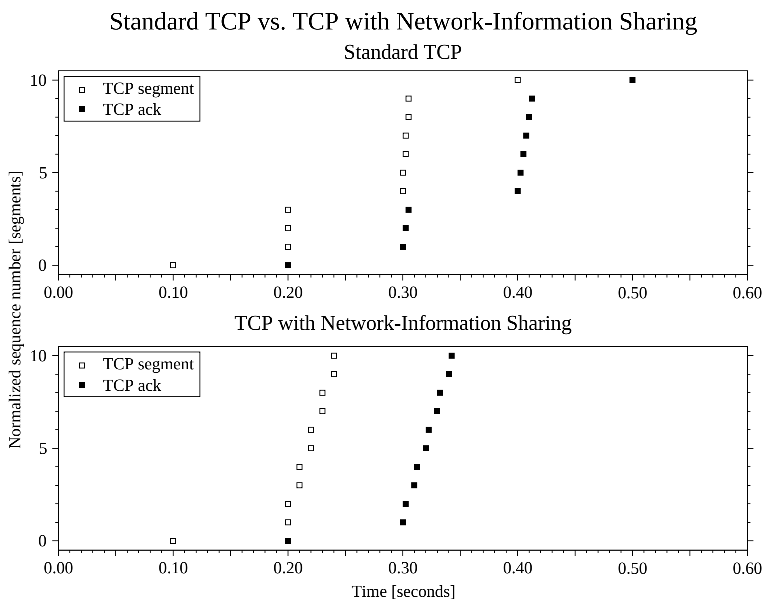


Figure 3.2: Potential of network-information sharing in an Internet end system

3.4 Network-Information Sharing between Internet Routers and End Systems

3.4.1 Introduction

Network information can be also shared between Internet routers and end systems, i.e., these approaches share network information which is distributed over the whole network. All of these NIS approaches are able to implicitly or explicitly send network information from the routers back to the senders (cf. Figure 3.3).

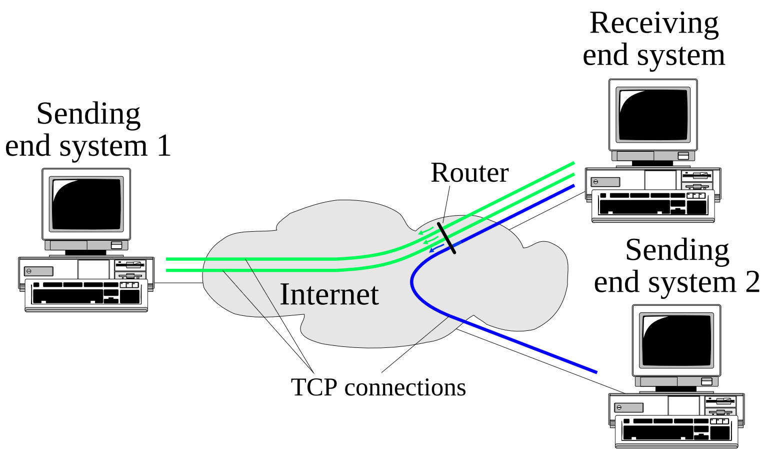


Figure 3.3: Network-information sharing between Internet routers and end systems

3.4.2 Router Congestion Feedback

This network information can include information about the current load in the routers, the load that can be currently accepted by the routers, and/or information about the current congestion status of the routers. Thus, some of these NIS approaches can be finer specified as router load feedback (RLF) or router congestion feedback (RCF) approaches. In the following, the abbreviation RCF is used for both variants of NIS approaches between routers and end systems as a synonym.

In the last chapter, two of such RCF approaches developed to increase the performance of TCP connections in case of congestion have been already described: RED and ECN. But these two mechanisms provide only a implicit or explicit single-bit congestion information (no congestion / congestion) as a router feedback for the TCP senders. Therefore, with these two NIS approaches a TCP sender is not able to adequately react on the current load conditions in the routers providing these simple RCF mechanisms. And since these two router feedback mechanisms are only designed for that TCP senders could faster react on (impending) congestion in a router, their performance gain is limited to this case. Since ECN and RED are not able to signal to the TCP senders that additional bandwidth is available in the network, the TCP senders are not able to faster increase their sending window as they can do with the standard TCP congestion controller.

There exist other RCF approaches that are able to finer control the sending window of TCP connections in the case of (impending) congestion and also in the case of additional available bandwidth in parts of the network. Some of these RCF approaches will be later described in detail (cf. Chapters 12 and 13). But in the following, the main tasks and design principles of such more sophisticated RCF approaches are generally described. In addition, an example is given that shows the potential benefit these RCF approaches can have compared to standard TCP.

3.4.2.1 Main Tasks

Three main tasks have to be performed by a RCF approach:

(1) **Calculation of feedback information:** Each router is equipped with a RCF controller that is able to measure the current load in the router and calculates feedback information according to this load information.

(2) **Transport of feedback information:** The feedback information calculated by the RCF controller in a router has to be transferred back to the senders.

(3) **Reaction on feedback information:** The senders receive the feedback information and adjust their sending window according to this feedback information.

3.4.2.2 Design Principles and Aspects

This section considers in detail the main design principles and aspects related to RCF approaches that share network information between Internet routers and end systems.

3.4.2.2.1 Direction of Network-Information Transfer

Some RCF approaches, e.g., RED and ECN, share network information only in one direction from the routers to the senders. Therefore, these RCF approaches are classified as one-way RCF approaches.

Other RCF approaches are based on a bi-directional network-information sharing between routers and senders. In the first direction, the TCP senders send network information, e.g., information about their current congestion-control status, to the routers. With these information and together with internal router status information, the routers are able to generate more appropriate congestion-control feedback information for each sender on a per-flow basis. This can be done without storing per-flow information inside the routers, a criterion that is necessary to develop a scalable RCF approach. In the second direction, this congestion-control feedback information is sent back from the routers to the senders. Such RCF approaches are classified as two-way RCF approaches.

3.4.2.2.2 Mechanisms to Transfer Network Information

If network information should be shared between end systems and routers it is important to consider how this can be done. The following list describes three possible mechanisms to implement such RCF approaches in the Internet:

(1) Network information can be transferred by using existing transport-protocol mechanisms of TCP. This is typically done in one-way RCF approaches where the congestion-control feedback information from routers is sent back to the TCP senders. These approaches have the advantage that they are transparent for the TCP instances in the end systems. But in general their expected performance gain is rather limited compared to the other two mechanisms.

(2) Network information can be also transferred encapsulated in additional protocol messages. These new protocol messages can be transferred piggy-back in standard protocol messages, e.g., as a new TCP header option, or in separate new PDUs. In addition, also the semantic of existing protocol messages can be changed to provide new RCF mechanisms. In all cases, this approach requires that the standard TCP algorithms in the end systems have to be (slightly) changed to support the new RCF functionalities. With this mechanism it is possible to provide both one-way and two-way RCF approaches with a much higher expected performance gain compared to RCF approaches based on mechanism (1).

(3) The most complex mechanism to implement a one-way or two-way RCF approach is to develop a complete new transport protocol, possibly based on TCP. It can be expected that the performance gain of such a RCF approach is at least as high as that of RCF approaches based on mechanism (2). But this advantage in terms of improving the performance is combined with huge problems regarding the deployability of such an approach in the current Internet.

3.4.2.2.3 Feedback Information

RCF approaches can implicitly or explicitly share network information between Internet routers and end systems. In addition, this network information can be shared once, i.e., at the setup of a new data stream, or continuous during the whole lifetime of a data stream.

Feedback information is used to control the load that senders can put into the network. Some RCF approaches are designed only to restrict this load while other RCF approaches have also the capability to increase this load much faster than it is possible with standard congestion-control mechanisms used in the current Internet.

3.4.2.2.4 Complexity

Another important design aspect of a RCF approach is its complexity. This includes the time and space consumption of the RCF controller located in the router(s) and of the possibly adapted congestion controller located in the end systems. In addition, also the protocol overhead that is necessary to transfer the shared network information between the end systems and the router(s) is of interest.

The following Table 3.2 summarizes the design principles and aspects of network-information sharing approaches between Internet routers and end systems.

Table 3.2: Design principles and aspects of network-information sharing approaches between Internet routers and end systems

Design principle / aspect	Possibilities
Network-information sharing direction	one-way / two-way
Transfer of network information	transparent / non-transparent
Feedback information	implicit / explicit, one-time / continuous
Complexity	time and space in routers / end systems, protocol overhead

3.4.3 Potential of Network-Information Sharing between Internet Routers and End Systems

The following Figure 3.4 shows a snapshot of a simple simulation scenario in which two TCP connections traverse a bottleneck router whose RCF-capabilities are either switched off or switched on. The senders of these two TCP connections are located in the end systems 0 and 1, their receivers are located in the end systems 4 and 5. The segments and acknowledgments of both TCP connections traverse the routers 2 and 3 in the network. Since all links in the network scenario have equal properties regarding the bandwidth (10 Mbps) and propagation delay (5 ms), router 2 is the bottleneck router in this network scenario. Its current queue length is also pictured in the figure. Both TCP connections start sending their segments into the network at the same time. If the RCF-capabilities in the bottleneck router are switched off, packet losses due to congestion in this router are likely to occur. This is the outcome of the uncontrolled grow of the queue in the bottleneck router up to its maximum queueing capacity of 19 packets. Since these packet losses are not equally shared among the two TCP connections over the simulated time, they obtain largely different mean throughputs of 102.00 and 496.00 segments per second during the simulated time.

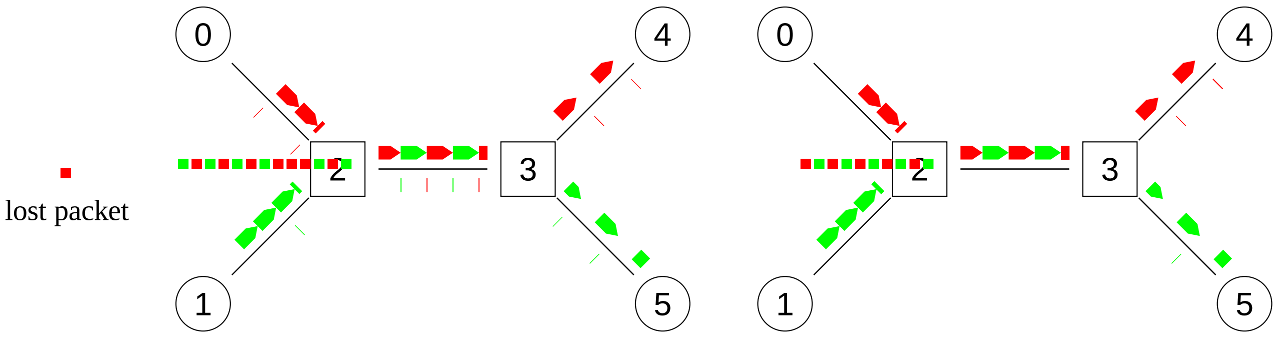


Figure 3.4: Potential of network-information sharing between Internet routers and end systems (simulation with non-RCF-capable routers on the left side, simulation with RCF-capable routers on the right side)

Thus, fairness between similar TCP connections cannot be guaranteed by the standard TCP congestion control as it is used in the current Internet.

But if the RCF-capabilities in the bottleneck router are enabled, the two TCP senders receive information about the load in this router that they can adjust their current sending window on this load information. Then, the queue length of the bottleneck router is accurately controlled that packet losses are very rare events. As a result, both TCP connections obtain comparable mean throughputs of 244.00 and 276.00 segments per second during the simulated time. It is obvious that these two TCP connections are much more fairly treated by the more sophisticated TCP congestion control based on network-information sharing between Internet routers and end systems compared to the standard TCP congestion control.

3.5 Summary

The first NIS type, network-information sharing between some TCP senders in a single end system, is able to improve the performance of those TCP connections of a single end system which have access to network information collected by other TCP connections. Thus, only some of the TCP connections in an end system can benefit from such a NIS approach. In addition, identifying TCP connections of an end system that can share network information is a difficult task in the real Internet.

The second NIS type, network-information sharing between routers and end systems, is able to improve the performance of those TCP connections which traverse RCF-capable routers in the network. Thus, dependent on the number and distribution of RCF-capable (bottleneck) routers in the network only a few up to all TCP connections in the network or in parts of the network can benefit from such a NIS approach.

For both NIS types in IP-based networks several existing and new approaches are considered in detail in this dissertation regarding their functionalities, properties, complexity, applicability in the current and future Internet, and a priori expected and later in different network scenarios evaluated performance gain.

3.6 Performance Evaluation of Network-Information Sharing Approaches

3.6.1 Introduction

In this dissertation, the performance of several NIS approaches compared to standard TCP is evaluated by simulations and measurements. This section briefly describes the most important metrics used for this performance evaluation. In addition, the general simulation framework is portrayed which is the basis for the derived simulation models that are used to evaluate the performance of the different NIS approaches compared to standard TCP.

3.6.2 Performance Metrics

The main metrics used to evaluate the performance of different NIS approaches compared to standard TCP are the throughput of TCP connections and the fairness between TCP connections (cf. Section 2.3.2.7). In addition, also the queue length in the bottleneck routers and the number of packet losses or the packet loss rate, respectively, in these routers are of interest. These performance metrics are all objective metrics.

In the following, it is described how these performance metrics are used. More details about the calculation of these performance metrics in the simulations can be found in the Sections 7.2 and 15.2.

3.6.2.1 Throughput

The most important performance metric for an evaluation of NIS approaches is the throughput of TCP connections. Dependent on the type of the NIS approach that is evaluated, throughputs for the following TCP connection(s) are considered:

- The mean throughput of a new TCP connection entering an ensemble of TCP connections which can share network information. This throughput is calculated over the whole duration of the new TCP connection.
- The throughputs of concurrent TCP connections in an ensemble of TCP connections which can share network information. For each of these TCP connections its throughput is calculated during the time of concurrency of the TCP connections in an ensemble.
- The throughputs of TCP connections which traverse a router with optionally switched on RCF capabilities. For each of these TCP connections its throughput is calculated over its whole duration.

All these throughputs are calculated by counting the number of TCP segments sent *and* acknowledged during the measurement interval using Equation (2.16). Mean values for these throughputs are calculated using the two different mean throughput calculations defined in Equations (2.17) and (2.18) (cf. Section 2.3.2.7).

The first two mean throughputs are then used to evaluate the throughput performance of TCP connections which share network information compared to the mean throughputs of standard TCP connections.

The third mean throughput is used to evaluate the throughput performance of TCP connections which are assisted by RCF approaches compared to the mean throughput reached by standard TCP connections.

3.6.2.2 Fairness

For the first type of NIS approaches the fairness (index) between TCP connections which can share network information is an interesting and important performance metric. The fairness (index) between these TCP connections is expressed by comparing the throughputs the TCP connections reach during their time of concurrency according to the fairness-calculation defined in Equation (2.19) (cf. Section 2.3.2.7). A mean fairness (index) is then computed by using the mean fairness calculation defined in Equation (2.20) (cf. Section 2.3.2.7).

3.6.2.3 Packet Losses

The number of packet losses or the packet loss rate, respectively, in specific routers is used to compare different NIS approaches with standard TCP regarding their aggressiveness in putting packets into the network.

3.6.3 Simulation Framework

The following Figure 3.5 shows the basic composition of the simulation framework that is used to evaluate the performance of different NIS approaches compared to standard TCP. In this simulation frame-

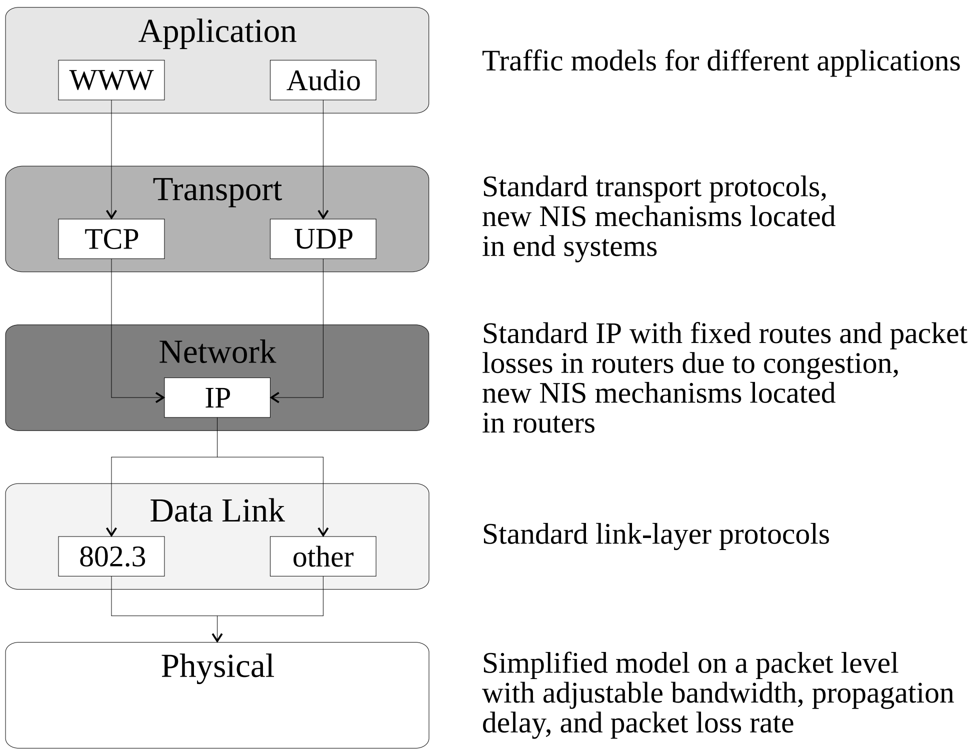

Application
WWW
Audio
Traffic models for different applications
Transport
TCP
UDP
Standard transport protocols, new NIS mechanisms located in end systems
Network
IP
Standard IP with fixed routes and packet losses in routers due to congestion, new NIS mechanisms located in routers
Data Link
802.3
other
Standard link-layer protocols
Physical
Simplified model on a packet level with adjustable bandwidth, propagation delay, and packet loss rate

Figure 3.5: Basic composition of the simulation framework

work, the upper layers of the Internet protocol stack down to the data-link layer are carefully designed to reflect the real properties of the Internet in the simulated network scenarios. But in order to reduce the complexity of the whole simulation framework and to limit the simulation time, the properties of the physical layer have to be expressed on an abstract packet-based level, hiding the very complex properties of real physical layers. In this dissertation, the simplification of the physical layer has only an influence on the simulation results if errors in (wireless) links are modeled.

In the application layer different traffic generators for WWW and audio (voice) data are used to generate the load in the simulated network. These application-based data is transported through the simulated network by using either TCP or UDP as the underlying transport protocol. In the network layer standard IP in its current version 4 is used as the network protocol. And in the data-link layer widely used point-to-point and LAN protocols are performed.

NIS approaches of the first type are located in the TCP instances running in the end systems. And NIS approaches of the second type are located in the routers. For NIS approaches of the second type which are not transparent for the standard TCP instances running in the end systems also some additional control mechanisms have to be included in the TCP instances.

Figure 3.6 shows a general network scenario that can be derived from the simulation framework. In

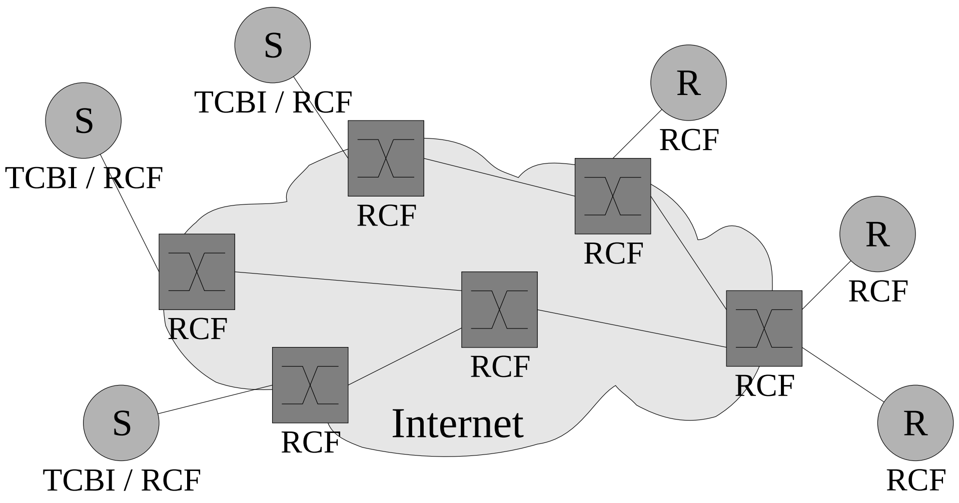


Figure 3.6: General network scenario derived from the simulation framework

the sending end systems (Ss) of the network an optionally TCBI approach can be installed to provide an improved congestion control based on network-information sharing between some TCP senders of an end system. In addition, the routers—and if required also the sending and/or the receiving end systems (Rs)—can be optionally equipped with RCF capabilities to provide an improved congestion control in the network based on network-information sharing between the routers and the end systems. Details about the network scenarios derived from the simulation framework and used for the simulations can be found in the Sections 7.3 and 15.3.

The whole simulation framework is implemented in the ns-2 [61] environment in its version 2.1b9. Details about the the implementations of the different simulation scenarios in the ns-2 environment can be found in the Sections 7.3.3 and 15.3.4.

Part I

Network-Information Sharing in Internet End Systems

Chapter 4

Network-Information Sharing in Internet End Systems: Related Work

4.1 Introduction

In this chapter, the most important and interesting NIS approaches in Internet end systems are described and compared with regard to their properties, functionalities, requirements, and expected performance gain. The focus of this chapter is on the general mechanisms of these NIS approaches and not on details of their (congestion-)control algorithms.

The remainder of this chapter is organized as follows. In Section 4.2, the passive network performance probing approach Shared Passive Network Performance Discovery (SPAND) [62] is described. Section 4.3 considers the common congestion-control approach Ensemble-TCP (E-TCP) [63]. And in Section 4.4, the main ideas of another common congestion-control approach, the Congestion Manager (CM) [64, 65], specifically designed for data streams of different applications in an end system are described. A summary of the main features and of the expected performance of these approaches can be found in Section 4.5.

4.2 Shared Passive Network Performance Discovery (SPAND)

4.2.1 Introduction

If it is possible to predict sufficiently well the network performance, e.g., the available bandwidth, the latency, or the packet loss probability, between a pair of end systems many applications could benefit from such an approach. For example, applications can choose the server with the best current network performance from a list of servers that offer the same service, e.g., FTP or WWW mirror sites.

Each of these applications should be able to determine in advance the in future expected network performance between pairs of end systems. Previous work in this research area (cf. [62]) has been mainly focused on active network-probing mechanisms from individual end systems. These active network probes are developed to measure, for example, the round trip time and the available peak or fair share bandwidth between two end systems by sending a burst of packets or some special ICMP echo packets from one end system to the relevant end system from which they are sent back to the sender. Depending on the queueing implementation in the routers in the network path between the end systems the time intervals between the received echoed packets can be used to calculate the whole bandwidth or the available fair share of the bandwidth in the bottleneck link in the network path [66, 67].

But active network-probing approaches have three main drawbacks (cf. [62]): First, these network-probing approaches inherently increase the network traffic. Sometimes this extra network traffic can become a measurable fraction of the whole traffic which must be handled by some of the servers. Second, active network-probing mechanisms require some time for their probes and the evaluation of the probe results. And third, network probing from a single end system results in less accurate network information and more redundant network probes from nearby end systems than a network-probing mechanism coordinately performed for a set of end systems.

The Shared Passive Network Performance Discovery (SPAND) [62] approach eliminates these disadvantages by using network-performance measurements from a collection of end systems combined with passive network-performance prediction algorithms. This approach can be used only if the network performance between two end systems is predictable precisely enough from earlier network-performance measurements done by nearby end systems and for a longer duration. Past work [68, 60] has shown that the network performance between two end systems, i.e., one client and one server, can be stable for several minutes and can be very similar to the network performance observed by other nearby clients.

The SPAND approach is mainly developed for end systems working as clients to choose the server with the best current network performance from a list of servers that offer the same requested service. In such a scenario, the collected network information is (re)used on the receiver side of the data streams. Although the original SPAND approach is not designed for network-information sharing in *sending* end systems, the main mechanisms used in SPAND to collect network information from some data streams and reuse this network information for other data streams, are nevertheless interesting and valuable for developing a network-information sharing approach in sending end systems based on a modified SPAND approach.

In the next three Sections 4.2.2 to 4.2.4, a general look at passive network-probing approaches and particular at the SPAND approach is taken. This consideration is done with regard to a network-information sharing approach located in sending end systems. Finally, Section 4.2.5 presents some performance results of SPAND.

4.2.2 Passive Network-Performance Probing

According to [62], there are two main reasons for the usage of passive instead of active network-performance probing mechanisms. First, active network probing mechanisms require additional network traffic for their network probes. For example, some of the existing active network-probing approaches require tens of kilobytes of probe traffic for only one meaningful probe answer. This amount of probe traffic is a significant fraction of or even exceeds the mean transfer size of typical TCP connections transferring WWW-based data. And second, an active probing-based decision which of the possible servers to use for the upcoming data transfer needs some time, since the network paths to all possible servers have to be probed one after another. A passive network-probing approach avoids these drawbacks.

But passive network probing mechanisms also have one main disadvantage. They can only collect network-performance information for a remote end system if one of the end systems equipped with passive network-probing mechanisms is connected to the remote end system. Thus, to avoid out-of-date network-performance information or no network-performance information at all for a local end system about remote end systems the local end system has to establish connections often enough to the remote end systems to be able to passively probe the current network performance.

In passive network-probing mechanisms the collection of network-performance information should take into account whether the network-performance measurements are done by TCP- or UDP-based data streams, since for different transport protocols the network-performance measurement results can vary and lead to inaccurate network-performance predictions for other transport protocols than for those which were used for the measurements.

4.2.3 Network-Information Sharing

In general, a single end system is not connected frequently enough to the remote end systems of interest to measure the current network performance in the network paths to the remote end systems often enough such that a passive network-performance probing approach can be reasonably used to predict the future network performance in these network paths sufficiently exactly. An alternative to that is the collection of the current network-performance measurements of many end systems located in a local Internet domain. This can be done in a centralized way by using a (network-)performance server which is regularly informed by the end systems about their current network-performance measurement results. Then, in the performance server a passive network-probing approach can be used to find the remote end systems that provide the best current network performance. The network information obtained by the passive network-probing approach in the performance server can be shared over the end systems in the local Internet domain, for example, by using a network-information request/response mechanism between the end systems and the performance server.

But network-information sharing should not be applied for end systems with different network connectivity, e.g., between end systems which are connected via modems to the Internet and end systems connected to the Internet via an Ethernet-based LAN, or between end systems with different protocol versions. In these cases, the shared network information might be too inaccurate to be beneficial (cf. [62]).

4.2.4 SPAND Prototype Implementation

The architecture of the SPAND prototype [62] is shown in the following Figure 4.1. It mainly consists of several clients with a modified network-protocol stack and one performance server. The additional so far not explained packet-capture host is only used for network-performance measurements of those data streams going to clients whose protocol instances are not able to perform network-performance measurements on their own. All clients, the performance server, and the packet-capture host form a local domain in the SPAND architecture.

The results of the network-performance measurements in the clients and in the packet-capture host

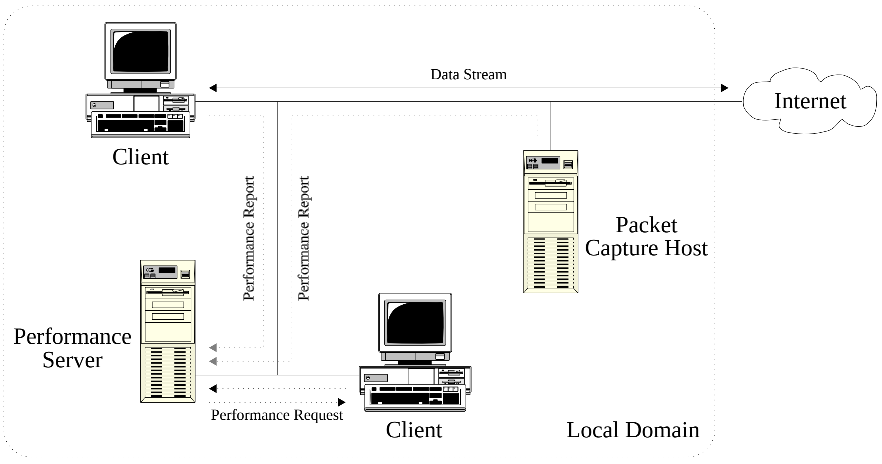


Figure 4.1: Architecture of the SPAND prototype

are sent as performance reports to the performance server of the local SPAND domain. In the performance server a passive probing mechanism is used to evaluate the performance reports and determine the current network performance in the network paths to the remote end systems of interest.

If, for example, a client wants to use a service offered by two or more remote end systems it first contacts the performance server of the local domain using a performance request. If the performance server has network-performance information about the remote end systems it informs the client accordingly. Now the client is able to choose the remote end system with the best current network performance. Otherwise, the client uses the default remote end system.

Depending on the number and frequency of performance reports received by the performance server, on the number of considered remote end systems, and on the runtime of the passive network-performance probing approach used in the performance server, the fraction of accurately answered performance requests might be high or low.

In the context of TCBI, the network-information sharing of the modified SPAND approach can be considered as an ensemble-TCBI one-time network-information sharing approach for TCP senders distributed over sending end systems in a subnetwork. TCBI-mechanisms are then performed on a LAN-to-host or a LAN-to-LAN basis and the TCP connections of a "SPAND-ensemble" are slightly coupled.

4.2.5 SPAND Performance Results

Measurements [62] in an industrial environment with the SPAND prototype have shown that the performance server is able to answer 70 % of the performance requests within receiving the first 300 performance reports, and after around 10000 received performance reports the performance server reaches a steady-state performance request answering rate of about 95 %. The measured accuracy of the answered performance requests shows that the predicted throughput of the performance server for a network path to a remote end system is in 69 % of the answered performance requests within a factor of 2 and in 90 %

of the answered performance requests within a factor of 4 of the real observed throughput (see [62]). The developers of SPAND hope that with some modifications in the performance-capture host and in the performance server they will be able to improve the accuracy of the SPAND performance estimations.

4.3 Ensemble TCP (E-TCP)

4.3.1 Introduction

One important feature of a common congestion control—if desired—is to avoid the drawbacks of the following well-known behavior of several concurrent TCP connections of an end system: Regarding the number of segments put into the network, N concurrent TCP connections of an end system are roughly N times as aggressive to the network than a single TCP connection is. This behavior of several concurrent TCP connections of an end system can lead to unfairness towards TCP connections of other end systems in the network. In addition, due to the aggressive behavior of parallel TCP connections the network can suddenly be highly congested. Then many packet losses occur with the result that some or all of the parallel TCP connections have to perform a fast retransmit or even a slow start such that the overall efficiency of the data transmission in the network will dramatically decrease.

One solution to that problem is the Ensemble TCP (E-TCP) [63] approach. In this common congestion-control approach for TCP connections a group of concurrent TCP connections with the same receiver or with receivers in the same subnet are combined to one ensemble. Together with their connection status, i.e., their TCP control variables, these TCP connections of an ensemble have to perform a common congestion control whose aggregated behavior and aggressiveness to the network is similar to the behavior and aggressiveness of a single TCP connection. Therefore, a single TCP connection of an ensemble is less or at most as aggressive as a single standard TCP connection.

In addition to this Ensemble TCP state sharing, the E-TCP approach provides a mechanism similar to the temporal-TCBI mechanism where the network information obtained from senders of previously closed TCP connections is reused for the sender of a new TCP connection to the same receiver or to a receiver in the same subnet. In this temporal-TCP state sharing the E-TCP approach can be regarded as an example implementation of the ideas behind the temporal-TCBI approach.

Except for some mechanisms supporting T/TCP [69] all changes in a given TCP implementation providing the E-TCP approach are located in the E-TCP-capable end system. Therefore, it is worthwhile to investigate the E-TCP approach as an example implementation of a common congestion-control algorithm for TCP connections in a sending end system. UDP data streams are not supported by the E-TCP approach. But with some minor extensions the E-TCP approach should be amenable to support also UDP data streams and implement a common congestion control for some data streams of a sending end system.

4.3.2 E-TCP Design and Implementation

TCP connections of an E-TCP-capable end system with the same receiving end system or with receiving end systems in the same subnet are combined to an ensemble. For each ensemble one ensemble control block (ECB) is created in which the necessary control variables and data structures for a common congestion control of the TCP connections in the ensemble are stored. The ECB consists of

- several TCP control variables aggregated from the original TCP control blocks (TCBs) of the TCP connections in the ensemble, e.g., the congestion window, the slow start threshold, the (smoothed) round trip time, and the variance in round trip time,
- some new data structures and additional parameters, e.g., the list of the TCBs of the TCP connections in an ensemble, the maximum segment size (MSS) supported in the network path between the (pair of) end systems related to the ensemble, and a single bit that specifies if rate-based pacing is used or not (with that mechanism the performance of restarted idle TCP connections can be improved [70]),
- a time-sorted list of previously sent but not yet acknowledged TCP segments of the TCP connections in the ensemble together with an associated skipped acknowledgment counter for each TCP segment in the list (this list is used for an adaptation of a TCP segment retransmission and congestion decreasing scheme originally designed for TCP-INT [71]),
- some other control parameters necessary only for T/TCP [69] connections which are not in the scope of this dissertation.

Two types of ECBs can be distinguished: active ECBs and cached ECBs. An active ECB has at least one active TCP connection associated with it. A cached ECB is not associated with a currently active TCP connection. It consists of ECB values from an ensemble where the last TCP connection of this ensemble has been recently closed. A new TCP connection to the same receiving end system or to a receiving end system in the same subnet related to such an cached ECB can benefit from the network information stored in this cached ECB, e.g., to shorten the transient start phase with a lower performance by using more adequate initial values of some of the TCP control variables. To avoid out-of-date network information stored in a cached ECB, the lifetime of cached network information should be limited and an ”aging“ algorithm for the network information stored in the cached ECB should be used.

Each ensemble uses a scheduler for TCP segments with four predefined segment priorities. This is done to give retransmitted TCP segments a higher priority over other TCP segments or to have the ability to share the whole currently possible sending window of an ensemble in different proportions between the TCP connections of the ensemble. One interesting aspect is that in the implementation of E-TCP described in [63] the TCP retransmission timeout timer is not aggregated, i.e., each sender of a TCP connection of an ensemble manages its own TCP retransmission timeout timer.

The E-TCP approach can be considered as an instantiation of the TCBI approach that performs temporal- and ensemble-TCBI network-information sharing combined with a common congestion control between TCP connections of an ensemble. Due to the jointly managed list of sent but not yet acknowledged list of segments in an ensemble, the TCP connections of an ensemble are stronger coupled.

4.3.3 E-TCP Performance Results

In [63], the E-TCP approach has been investigated under a specific scenario with HTTP/1.0-traffic [16, 18] and no background traffic in the network. The simulations show a 17 %–61 % higher throughput for Ensemble TCP state sharing of E-TCP compared to standard TCP. For temporal-TCP state sharing of E-TCP the throughput gain is about 10 %–27 % compared to standard TCP. The developers of E-TCP feel certain that the idea of ensemble- or temporal-TCP state sharing in the E-TCP approach is also beneficial under other traffic conditions in the network.

4.4 Congestion Manager (CM)

4.4.1 Introduction

The Congestion Manager (CM) [64, 65] provides a common congestion control for all data streams of an end system to one receiver or to receivers in the same subnet from an end-to-end perspective. The main design principles of the CM are selected to provide this common congestion control. But the CM is also developed—if desired—to put the applications in control that the applications are able to react on network information delivered by the CM controller. Then, the CM supports heterogeneous data streams and applications of an end system.

Most of the mechanisms of the CM need no additional information from the network beyond the existing mechanisms of the used transport protocols or the applications. But for some mechanisms of the CM an adaptation of the protocol stack in the receivers is necessary to support a new lightweight information-feedback protocol, the CM protocol.

Although the CM approach is neither transparent for the applications running on an end system and using the cobgestion-manager framework nor is it strictly located only on the sender side of the data streams, it is valuable to investigate the main ideas and mechanisms of the CM with regard to develop a transparent common congestion control for TCP connections of a sending end system. Similar to E-TCP, TCP connections of an ensemble are jointly controlled by the CM that their aggregated aggressiveness to the network is comparable to that of a single standard TCP connection.

In the next Section 4.4.2, the architecture of the CM framework is described. In addition, some performance results obtained by investigations of a CM prototype are presented in Section 4.4.3.

4.4.2 CM Framework

In the following Figure 4.2 the structure of the new protocol stack of an end system using the CM is shown. The core of the CM framework is the Congestion Manager itself. It collects network information and maintains network statistics which are used to adapt the window or rate of the data streams of an end system following robust control principles, e.g., the additive increase multiplicative decrease (AIMD) approach [35]. In addition, the CM coordinates host- and domain-specific path information between the different data streams. The obtained path properties are shared between the data streams, since the transport protocols (TCP, UDP, and RTP) and some applications using the CM framework directly perform data transmissions only with the consent of the CM. Internally, the CM consists of a congestion

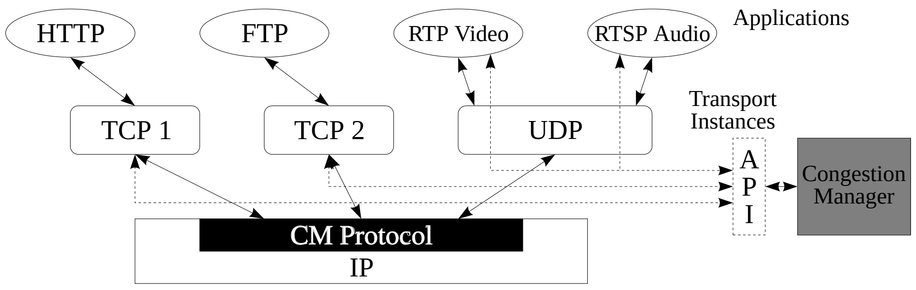


Figure 4.2: Structure of the new protocol stack of an end system using the Congestion Manager

controller, a scheduler, and a new protocol. The main features of these CM parts are described in the following (cf. [64, 65]):

- The congestion controller of the CM is implemented as a hybrid window- and rate-based scheme. A window-based AIMD mechanism similar to the existing TCP congestion-control algorithm NewReno is used to control the overall data transmissions of an end system dependent on the available bandwidth in the network. To reduce bursts of data transmissions of an end system, a traffic shaper based on a rate estimation combining the current congestion window and the smoothed round trip time is included in the congestion controller. If there is no feedback from the network, the congestion controller changes to an exponentially decaying rate-based scheme as a fallback procedure.

- A scheduler is used to share the available bandwidth between the existing data streams of an end system. With that approach fairness between data streams of an end system with equal properties as well as priority mechanisms between data streams of an end system with different properties can be provided.

- Two forms of feedback from the receivers are used in the CM to adjust its congestion window with response to packet losses due to congestion in the network: application notification (AN) and explicit feedback (EF). AN occurs when the applications or the transport protocols are able to send their own feedback information, e.g., packet losses or ECN from the routers, back to the sender. EF is used for those applications or transport protocols which do not have an integrated feedback mechanism. In this case, an explicit probing protocol from the CM to the receivers is used to obtain periodically feedback information, e.g., to detect packet losses or obtain other useful information, from the data streams of an end system on a per stream basis. This probing protocol should not require too much additional bandwidth in the network and should also be resilient to losses of probing-protocol messages. The frequency of sending probing-protocol messages is determined to twice per round trip delay. Due to congestion in the network, probing-protocol messages or responses can be lost, i.e., the sender has no accurate estimation of the current network state during these times of congestion. In this case, an exponential aging algorithm of the current network state is used that halves the sending rate of all data streams in an end system every so far measured minimum round trip time of the data streams during the duration of activity. This half-life period of the network state is selected as a compromise between a too conservative and a too aggressive behavior of the data streams of an end system to the network.

The transport protocols and some applications access the features of the CM by using a new Application Programming Interface (API) incorporated with a CM protocol located between the IP layer and the higher layers of the Internet protocol stack in an end system supporting the CM. The design of this new API is intended to enable the transport protocols and applications to adapt their sending rate to the current available bandwidth in the network paths to the receivers. Therefore, the API supports mechanisms to query the path status, schedule data transmissions, and update variables dependent on successful transmissions or congestion in the network. In addition, the API includes callback functionalities from the CM to the applications so that the applications can learn about the current network conditions and have the ability to decide which information how to transmit and what relative fraction of the available bandwidth is used for a data stream. This feature, which is motivated by the end-to-end argument and the principle of application level framing (ALF), reduces the possibility to have congestion in the network paths to the related receivers. The CM can also learn from the applications, since the API provides a callback mechanism from the applications to the CM to update the network-status parameters in the CM.

The CM protocol is designed to take advantage of all features of the CM, e.g., the probing protocol to periodically obtain feedback information from the receivers. However, a CM sender is able to communicate with a receiver that does not support the CM protocol and the CM. In this case, the CM in the sender works well only for those transport protocols and applications which provide feedback information to the sender. Whether a receiver supports the CM protocol and the CM or not can be determined by the CM sender with a two-way handshake protocol.

If the receiver supports the CM protocol and the CM, then every packet of the data streams of the sender is encapsulated in a CM protocol data unit with a CM protocol header. This CM protocol header consists of a protocol field which contains the original protocol type identifier of the encapsulated packet. With the type field in the CM protocol header the different types of CM protocol messages, e.g., a probe or a probe response message, can be chosen. In addition, each CM protocol header is marked by a flow ID that uniquely identifies the flow for which the CM protocol message is sent and a sequence number that reflects the incremental sequence number of the encapsulated packet in the flow.

In the context of TCBI, the CM approach contains a TCBI controller that performs one-time network-information sharing and common congestion control for TCP connections of an ensemble on a host-to-host basis. But the CM is also able to support congestion control for non-TCP data streams on the basis of network-probing mechanisms. In addition, applications running on top of the CM can be provided with congestion-control information by the CM that they can react on changed network conditions.

4.4.3 CM Performance Results

In [64], an experiment is described that compares the throughput of four independent TCP NewReno connections with the throughput of four coupled TCP/CM connections. It shows the expected result that the overall throughput of the single TCP NewReno connections is 15 % higher than the overall throughput of the TCP/CM connections. But while the throughputs of TCP NewReno connections vary by a factor of 2.7 all TCP/CM connections have nearly the same throughput. Therefore, for TCP connections the CM improves the fairness and the performance consistency between TCP connections that the predictability of the throughput of a TCP/CM connection is much better than for a TCP NewReno connection.

Another experiment [64] with a single TCP/CM connection and an audio data stream shows that the audio application is able to react to changing network conditions by choosing encoding algorithms with different coding rates that match the current throughput of the TCP/CM connection. The overall throughput of the TCP/CM connection and the audio data stream is equal to the throughput of a TCP NewReno connection.

4.5 Summary

In the previous sections, the main ideas of three existing one-time network-information sharing or common congestion-control approaches are described. The following Table 4.1 summarizes the main functionalities and properties and the expected performance gain compared to standard TCP of these approaches.

Table 4.1: Overview of the three existing network-information sharing approaches

	SPAND	E-TCP	CM
Ensemble-decision criterion	IP addresses		
Ensemble-creation	static		
performs ensemble / temporal one-time network-information sharing	yes / no	yes / yes	yes / yes
performs common congestion control	no	yes	yes
supports TCP / UDP	yes / yes	yes / no	yes / yes
is located only in sending end systems	(yes)	yes	no
requires additional network traffic (e.g., probe packets)	yes	no	no
is transparent for applications	yes	yes	(no[1])
complexity of the implementation	high	medium	high
fairness gain of concurrent TCP connections (expected[2])	+	++	++
throughput gain of a new TCP connection (expected[2])	+	++[3]	++[3]
throughput gain of concurrent TCP connections (expected[2])	+	-[4]	-[4]

[1]Transparency can be reached if (1) standard socket calls are translated to CM-specific calls and if (2) no application notification is used

[2]Compared to standard TCP

[3]If cached network information are used

[4]Due to the selected strategy that all TCP connections of an ensemble should be as aggressive to the network as a single TCP connection

For comparison reasons, it is assumed that the functionalities of the original receiver-side located SPAND approach are adapted to a sender-side located one-time network-information sharing approach similar to the two other approaches. The overview shows that the E-TCP approach has some advantages over the other three approaches, except for the last considered property where the restricted aggressiveness of E-TCP will prevent a better result. The CM provides more functionalities than E-TCP, e.g., the mechanism that applications can adapt their sending rate on the current network conditions, but at the cost of a much higher complexity of the implementation and a new API which changes the standard transport-layer protocol interfaces used in current Internet end systems.

Chapter 5

Network-Information Sharing in Internet End Systems: New Approaches

5.1 Introduction

The previous chapter has shown that network-information sharing in a single or in multiple end systems can be done in different ways. In this chapter, two new network-information sharing approaches are described in detail. The first approach, called Fair Share TCBI (FS-TCBI), allows a one-time network-information sharing between senders of existing TCP connections of an ensemble and the sender of a new TCP connection entering this ensemble, i.e., an ensemble-TCBI one-time network-information sharing is performed. Then, the sender of a new TCP connection is able to start with more accurate initial values for some of its TCP control variables than it is intended by the standard. The second approach, called Ensemble Flow Congestion Management (EFCM), extends FS-TCBI to a common congestion control between senders of TCP connections of an ensemble where network information is continuously shared between all senders of TCP connections of an ensemble. Regarding the maximum number of outstanding segments, FS-TCBI and EFCM are developed to be at most as or as aggressive as standard TCP connections (cf. Chapter 6).

Although FS-TCBI performs a subset of EFCM-mechanisms, there are some differences in the details of their algorithms. Both approaches perform identical one-time network-information sharing algorithms between a new TCP connection entering an ensemble and the existing TCP connection in this ensemble regarding the setting of the control variables (cf. Sections 5.2.5.1 and 5.3.6.1), but their pacing algorithms (cf. Sections 5.2.5.2 and 5.3.6.3) differ in this case. In addition, the FS-TCBI controller does not redistribute the values of the CWND and SSTHRESH of a recently closed TCP connection leaving the ensemble among the remaining TCP connections in that ensemble. Also the complexity of both approaches is different (cf. Sections 5.2.6 and 5.3.7). Therefore, for a better understanding, both new network-information sharing approaches are separately described in detail in their own Sections 5.2 and 5.3. But their main properties and functionalities, complexity, and expected performance gain compared to standard TCP are summarized in Section 5.4. Finally, the practicability of FS-TCBI and EFCM in real Internet end systems is considered in Section 5.5.

5.2 Fair Share TCP Control Block Interdependence (FS-TCBI)

5.2.1 Introduction

FS-TCBI is a one-time network-information sharing approach based on TCBI ideas. In this section, the design constraints, main properties, functionalities, and algorithms of FS-TCBI are explained in detail. In addition, a rough estimate of the additional time and space consumption of the EFCM controller compared to standard TCP is given.

5.2.2 FS-TCBI Design Constraints and Properties

The design constraints and properties of the FS-TCBI approach are:

- The FS-TCBI control algorithms are part of the transport layer of a sending end system, i.e., an end system where the TCP senders of the originating applications are located. Neither intermediate network nodes nor the receiving end system(s) have to be modified.
- FS-TCBI supports standard transport-layer interfaces, i.e., sockets. Therefore, FS-TCBI is transparent for all applications and Internet services running on an end system with FS-TCBI-capabilities.
- The algorithms of the FS-TCBI controller ensure that an ensemble of n TCP connections is no more aggressive to the network than n separately controlled TCP connections of the end system.
- If a TCP connection enters an ensemble, information about available bandwidth should be fairly shared among senders of all TCP connections in this ensemble.

5.2.3 FS-TCBI Functionalities

The FS-TCBI controller performs one-time network-information sharing for the sender of a new TCP connection. It does not, however, reuse network information obtained from senders of recently closed TCP connections (as, e.g., temporal-TCBI [57] does). It also does not support a TCP-friendly behavior of UDP flows by using network information obtained from senders of existing or recently closed TCP connections.

To avoid a bursty sending behavior of the senders of FS-TCBI-controlled TCP connections, the FS-TCBI controller uses a rate-based pacing mechanism for consecutive TCP segments sent by each sender.

5.2.4 FS-TCBI Jointly Controlled TCP Control Variables

According to Section 2.3.2, TCP uses the following control variables for congestion and error control: congestion window (CWND), slow start threshold (SSTHRESH), smoothed round trip time (SRTT), and round trip time variance (RTTVAR). The first two TCP control variables restrict the load the sender of a single TCP connection can send into the network. The last three control variables lead to adequate retransmission timeout timer (RTO) values for TCP segments sent from the sender of a single TCP connection.

The FS-TCBI controller is able to share network information stored in the congestion windows, the slow start thresholds, the smoothed round trip times, and the round trip time variances of senders of TCP connections in an ensemble. Hence, the FS-TCBI controller restricts the load the sender of a new TCP connection entering an ensemble can send into the network. In addition, the sender of a new TCP connection entering an ensemble is able to use an adequate retransmission timeout timer even for the first TCP segment to send.

5.2.5 FS-TCBI Control Algorithms

In this section, the algorithms of the FS-TCBI controller for initializing the selected control variables of the sender of a new TCP connection entering an ensemble are described. Afterwards, the rate-based pacing mechanism of FS-TCBI is explained.

5.2.5.1 FS-TCBI Network-Information Sharing for a new TCP Connection

If network information is available for the sender of a new TCP connection, the sender of a new TCP connection will reuse this network information and will start with more adequate values for the load the network can cope with and for the retransmission timeout timer. How the FS-TCBI controller shares available network information is described in the following (cf. Section 5.3.6.1 and Figure 5.1):

Congestion window: The FS-TCBI controller determines the sum of all current congestion windows of the senders of existing TCP connections of the ensemble plus the standard initial (congestion) window IW $= 3$ (segments), representing the sender of a new TCP connection. This aggregated congestion window is used to calculate a fair share, i.e., an arithmetic mean value, of the aggregated congestion window for all senders of TCP connections in the ensemble. At the beginning of a new TCP connection, all senders of the TCP connections of the ensemble get this congestion window fair share as their new congestion window.

Slow start threshold: The FS-TCBI controller determines the sum of all current slow start thresholds of the senders of existing TCP connections of the ensemble plus the standard initial slow start threshold $44 = \lfloor 65535/1460 \rfloor$ (segments) in the case of a MSS of 1460 bytes, representing the sender of a new TCP connection. This aggregated slow start threshold is used to calculate a fair share of the aggregated slow start threshold for all senders of TCP connections in the ensemble. At the beginning of a new TCP connection, all senders of TCP connections of the ensemble are assigned this slow start threshold fair share as their new slow start threshold.

Smoothed round trip time: The FS-TCBI controller uses the current value of an aggregated smoothed round trip time of the senders of existing TCP connections of an ensemble as the initial smoothed round trip time for the sender of a new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial smoothed round trip time of the sender of the new TCP connection is set to the standard value.

Round trip time variance: The FS-TCBI controller uses the current value of an aggregated round trip time variance of the senders of existing TCP connections of an ensemble as the initial round trip time variance for the sender of a new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial round trip time variance of the sender of the new TCP connection is set to the standard value.

If a TCP connection leaves the ensemble, i.e., the TCP connection has been closed, its congestion window and slow start threshold are not redistributed among the senders of the remaining TCP connections in the ensemble as it is done, for example, in the later explained EFCM common congestion controller.

5.2.5.2 FS-TCBI Pacing Mechanism

In order to avoid a bursty sending behavior of the sender of a new FS-TCBI-controlled TCP connection, a pacing mechanism is implemented in the FS-TCBI controller. This pacing mechanism works as follows: The sender of every TCP connection in an ensemble can send at most two TCP segments in a burst. The time Δt between two consecutive segment bursts of the sender of a TCP connection is calculated by using the smoothed round trip time and the congestion window of the sender of a TCP connection:

$$\Delta t = \alpha \cdot \text{SRTT}(T_k)/\text{CWND}(T_k)$$

In the current version of the FS-TCBI controller the pacing factor α is set to the fixed value 2. If for a single TCP sender a pacing timer with an expiration time Δt is started, it is not updated by interim changed values of its smoothed round trip time and its congestion window. Thus, in some cases where its congestion window tends to increase often during the pacing time Δt, e.g., this is more likely to happen for lower round trip times, the pacing algorithm might be too conservative in sending new TCP segments compared to standard TCP. In future versions of FS-TCBI, this pacing mechanism could be improved if the pacing factor α is adapted dependent on the current smoothed round trip time, e.g., $\alpha = f(\text{SRTT}(T_k))$ using a monotonic increasing function f with a range starting from 1 for lower smoothed round trip times and growing to 2 for larger smoothed round trip times.

Remarks: In some specific cases, e.g., after the reception of a cumulative TCP acknowledgment (cf. Figure 2.12 at time 27674.38 s), a standard TCP sender is allowed to send more than two or its initial number of TCP segments in a burst. Even in these cases the pacing algorithm is performed for senders of FS-TCBI-controlled TCP connections. The reason for this conservative decision is that it is hardly possible to distinguish between all cases that can lead to such a bursty sending behavior of a TCP sender in a real end system without largely increasing the complexity of the FS-TCBI controller.

Evidently, the FS-TCBI computations impose some overhead in time and space. A rough estimate of the additional time and space complexity of the FS-TCBI controller compared to standard TCP is given in the following Section 5.2.6.

5.2.6 Complexity of the FS-TCBI Controller

The complexity of a controller can be expressed using the O-calculus [72, 73] that gives an asymptotic upper bound of a complexity metric, e.g., the time or space consumption of an algorithm or a controller.

Here, the time and space complexity of the FS-TCBI controller is compared to standard TCP by using an implementation of the FS-TCBI controller which is optimized for time consumption.

Let v be the number of shared TCP control variables of the FS-TCBI controller. For v_{fs} of these TCP control variables a fair share calculation, e.g., for the congestion window or the slow start threshold, and for the remaining v_{w} TCP control variables a weighted calculation, e.g., for the smoothed round trip time and the round trip time variance, is used. Let n be the number of concurrent TCP connections which form an ensemble. If a new TCP connection enters an ensemble or a recently closed TCP connection leaves an ensemble, the additional time consumption of the FS-TCBI controller compared to standard TCP can be estimated as:

$$\begin{array}{lcl} \text{\# Additions } (+,-) & = & O(1) \\ \text{\# Multiplications } (\cdot,/) & = & O(1) \\ \text{\# Assignments} & = & O(n) \end{array}$$

The considered implementation of the FS-TCBI controller has an additional space consumption compared to standard TCP whose upper bound is in $O(n \cdot v_{\text{fs}} + v_{\text{w}})$ per ensemble. If an FS-TCBI controller has to manage e ensembles at the same time, the upper bound of the additional space consumption is in $O(e + \sum_{i=1}^{e} n_i \cdot v_{\text{fs},i} + v_{\text{w},i})$; the additional e in the equation represents the space consumption of the search list that is used to determine whether a new TCP connection can join an existing ensemble or not. The overall additional time and space complexity of the FS-TCBI controller compared to standard TCP is low. And this additional complexity, except for the search list, is only needed if the FS-TCBI controller has to manage TCP connections in an ensemble.

5.3 Ensemble Flow Congestion Management (EFCM)

5.3.1 Introduction

EFCM is a common congestion-control approach based on TCBI ideas. In this section, the design constraints, main properties, functionalities, and algorithms of EFCM are explained in detail. In addition, a rough estimate of the additional time and space consumption of the EFCM controller compared to standard TCP is given. But first, the intention to develop a new common congestion controller is explained.

5.3.2 Intention of EFCM

In addition to the creation of an ensemble of TCP connections, a common congestion controller's job regarding the senders of the TCP connections in an ensemble can be divided into two main tasks: First, the common congestion controller has to manage the one-time network information sharing between senders of *existing* (or *recently closed*) TCP connections of an ensemble and the sender of a *new* TCP connection joining this particular ensemble. This task is similar to the controller's job in existing pure one-time network-information sharing approaches like the ensemble- or temporal-TCBI [57, 74, 75] or FS-TCBI described in the previous section. Second, the common congestion controller is responsible for the continuous network-information sharing between senders of *concurrent* TCP connections of an ensemble to reach a common congestion control for this ensemble.

These two tasks can be fulfilled in a number of different ways. In the last chapter, the main ideas, methods, and properties of the three most relevant approaches, e.g., Congestion Manager (CM) [65] and Ensemble-TCP (E-TCP) [63] have been described. One of these approaches, E-TCP, has been identified as a good basis for a new common congestion control approach called Ensemble Flow Congestion Management (EFCM). The E-TCP approach provides a common congestion control for TCP connections of an end system in a way that an ensemble of n TCP connections is no more aggressive to the network than a single TCP connection. Network information is shared between the sender of a new TCP connection and senders of existing or recently closed TCP connections and between senders of concurrent TCP connections. In addition, for every ensemble the E-TCP approach uses a scheduler that determines which sender of a TCP connection of an ensemble sends the next segment. A rate-based pacing mechanism can be optionally used for senders of TCP connections of an ensemble. The assumption that an entire ensemble should be at most as aggressive as a single TCP connection appears very conservative; it is here that EFCM and E-TCP differ most (details are presented later in this chapter). It can be claimed that EFCM results in considerable performance gains at a moderate increase in implementation complexity. The adverse effects on the throughput of other data flows in the network are negligible. This property of EFCM will be analytically shown in the next Chapter 6.

5.3.3 EFCM Design Constraints and Properties

The design constraints and properties of the EFCM approach are:

- The EFCM control algorithms are part of the transport layer of a sending end system, i.e., an end system where the TCP senders of the originating applications are located. Neither intermediate network nodes nor the receiving end system(s) have to be modified.

- EFCM supports standard transport-layer interfaces, i.e., sockets. Therefore, EFCM is transparent for all applications and Internet services running on an end system with EFCM-capabilities.

- The algorithms of the EFCM controller ensure that an ensemble of n TCP connections is no more aggressive to the network than n separately controlled TCP connections of the end system (see Chapter 6).

- Information about available bandwidth should be fairly shared among all TCP connections in an ensemble.

5.3.4 EFCM Functionalities

The EFCM controller performs one-time network-information sharing for the sender of a new TCP connection as well as common congestion control between senders of concurrently existing TCP connections of an ensemble. It does not, however, reuse network information obtained from senders of recently closed TCP connections (as, e.g., temporal-TCBI [57] does). It also does not support a TCP-friendly behavior of UDP flows by using network information obtained from senders of existing or recently closed TCP connections.

To avoid a bursty sending behavior of the senders of EFCM-controlled TCP connections, the EFCM controller uses a rate-based pacing mechanism for consecutive TCP segments sent by each sender. In addition, the EFCM controller is equipped with a joint ack clocking mechanism for every ensemble of TCP connections. This Ensemble Ack Clocking (EAC), (see Figure 5.1) allows to fairly partition the sent but currently not acknowledged TCP segments, i.e., the on-the-fly TCP segments, between the senders of TCP connections in an ensemble.

Not only, the EFCM controller jointly controls TCP connections in an ensemble by manipulating the control variables of the senders of these TCP connections and by calling some existing internal functions of the TCP implementation. The EFCM controller does not need any scheduler or other complex functionalities per ensemble.

5.3.5 EFCM Jointly Controlled TCP Control Variables

According to Section 2.3.2, TCP uses the following control variables for congestion and error control: congestion window (CWND), slow start threshold (SSTHRESH), smoothed round trip time (SRTT), and round trip time variance (RTTVAR). The first two TCP control variables restrict the load the sender of a single TCP connection can send into the network. The last three variables lead to adequate retransmission timeout timer (RTO) values for TCP segments sent from the sender of a single TCP connection.

The EFCM controller jointly controls the congestion window, the slow start threshold, the smoothed round trip time, and the round trip time variance of senders of TCP connections in an ensemble. Hence, the EFCM controller restricts the load an ensemble can send into the network and all senders of TCP connections of an ensemble share the same values for computing their own retransmission timeout timer.

5.3.6 EFCM Control Algorithms

In this section, the algorithms of the EFCM controller for initializing the selected control variables of the sender of a new TCP connection entering an ensemble and for updating the control variables in the senders of concurrent TCP connections in an ensemble are described. Afterwards, the rate-based pacing mechanism of EFCM is explained.

5.3.6.1 EFCM Network-Information Sharing for a new TCP Connection

If network information is available for the sender of a new TCP connection, the sender of a new TCP connection will reuse this network information and will start with more adequate values for the load the network can cope with and for the retransmission timeout timer. How the EFCM controller shares available network information is described in the following (see Figure 5.1, cf. Section 5.2.5.1):

Congestion window: The EFCM controller determines the sum of all current congestion windows of the senders of existing TCP connections of the ensemble plus the standard initial (congestion) window IW $= 3$ (segments), representing the sender of a new TCP connection. This aggregated congestion window is used to calculate a fair share, i.e., an arithmetic mean value, of the aggregated congestion window for senders of all TCP connections in the ensemble. At the beginning of a new

The values $T_0, T_1, \ldots$ indicate consecutive points in time, i.e., $T_0 < T_1 < \ldots$, where events occur which have an effect on jointly controlled TCP variables of an ensemble. These ensemble events are: a TCP connection enters or leaves the ensemble, a new or duplicate acknowledgment arrives at the sender of one of the TCP connections of the ensemble, or a timeout of the retransmission timer occurs in the sender of one of the TCP connections of the ensemble. Here, T_k is the time when the current ensemble event occurs, T_{k-1} is the last point in time where the jointly controlled TCP variables in the ensemble have been updated due to an earlier ensemble event. The unit for CWND and SSTHRESH is segments with a maximum possible segment size and not bytes.

- Network information reuse (performed by FS-TCBI and EFCM):
  The ensemble consists of $n-1$ TCP connections and one new TCP connection enters this ensemble:

$$\begin{aligned}\text{CWND\_AGG}(T_k) &= \text{IW} + \sum_{i=1}^{n-1} \text{CWND}_i(T_{k-1}) \\ \forall i : 1 \leq i \leq n : \text{CWND}_i(T_k) &= \frac{\text{CWND\_AGG}(T_k)}{n}\end{aligned}$$

$$\begin{aligned}\text{SSTHRESH\_AGG}(T_k) &= \text{Initial SSTHRESH} + \sum_{i=1}^{n-1} \text{SSTHRESH}_i(T_{k-1}) \\ \forall i : 1 \leq i \leq n : \text{SSTHRESH}_i(T_k) &= \frac{\text{SSTHRESH\_AGG}(T_k)}{n}\end{aligned}$$

$$\text{SRTT}_n(T_k) = \text{SRTT\_AGG}(T_{k-1})$$

$$\text{RTTVAR}_n(T_k) = \text{RTTVAR\_AGG}(T_{k-1})$$

- Common congestion control (only performed by EFCM):
  The ensemble consists of n TCP connections and TCP connection j either receives an acknowledgment or detects a packet loss:

$$\Delta\text{CWND}_j(T_k) = \begin{cases} 1 & \text{if an ACK is received, SS} \\ \frac{1}{\text{CWND}_j(T_{k-1})} & \text{if an ACK is received, CA} \\ 1 - \text{CWND}_j(T_{k-1}) & \text{a packet loss is detected} \end{cases}$$

$$\begin{aligned}\text{CWND\_AGG}(T_k) &= \text{CWND\_AGG}(T_k) + \Delta\text{CWND}_j(T_k) \\ \forall i : 1 \leq i \leq n : \text{CWND}_i(T_k) &= \begin{cases} \text{CWND\_AGG}(T_k)/n & \Delta\text{CWND}_j(T_k) < 0 \\ \text{CWND}_i(T_{k-1}) & \Delta\text{CWND}_j(T_k) > 0, i \neq l \\ \text{CWND}_i(T_{k-1}) + \Delta\text{CWND}_j(T_k) & \Delta\text{CWND}_j(T_k) > 0, i = l \end{cases}\end{aligned}$$

with a cyclical chosen l, $1 \leq l \leq n$, that $\forall i : 1 \leq i \leq n : \text{CWND}_l(T_{k-1}) \leq \text{CWND}_i(T_{k-1})$ (EAC)

$$\text{SSTHRESH}_j(T_k) = \begin{cases} \text{SSTHRESH}_j(T_{k-1}) & \text{if an ACK is received} \\ \max\{\text{CWND}_j(T_{k-1})/2, 2\} & \text{a packet loss is detected} \end{cases}$$

$$\begin{aligned}\text{SSTHRESH\_AGG}(T_k) &= \text{SSTHRESH}_j(T_k) + \sum_{1 \leq i \leq n, i \neq j} \text{SSTHRESH}_i(T_{k-1}) \\ \forall i : 1 \leq i \leq n : \text{SSTHRESH}_i(T_k) &= \frac{\text{SSTHRESH\_AGG}(T_k)}{n}\end{aligned}$$

$$\begin{aligned}\text{SRTT\_AGG}(T_k) &= \frac{n-1}{n} \cdot \text{SRTT\_AGG}(T_{k-1}) + \frac{1}{n} \cdot \text{SRTT}_j(T_k) \\ \forall i : 1 \leq i \leq n : \text{SRTT}_i(T_k) &= \text{SRTT\_AGG}(T_k)\end{aligned}$$

$$\begin{aligned}\text{RTTVAR\_AGG}(T_k) &= \frac{n-1}{n} \cdot \text{RTTVAR\_AGG}(T_{k-1}) + \frac{1}{n} \cdot \text{RTTVAR}_j(T_k) \\ \forall i : 1 \leq i \leq n : \text{RTTVAR}_i(T_k) &= \text{RTTVAR\_AGG}(T_k)\end{aligned}$$

Figure 5.1: Algorithms of the EFCM controller (fast retransmit and fast recovery are performed but due to simplicity not considered in these formulas)

TCP connection, all senders of TCP connections of the ensemble get this congestion window fair share as their new congestion window.

Slow start threshold: The EFCM controller determines the sum of all current slow start thresholds of the senders of existing TCP connections of the ensemble plus the standard initial slow start threshold $44 = \lfloor 65535/1460 \rfloor$ (segments) in the case of a MSS of 1460 bytes, representing the sender of a new TCP connection. This aggregated slow start threshold is used to calculate a fair share of the aggregated slow start threshold for senders of all TCP connections in the ensemble. At the beginning of a new TCP connection, all senders of TCP connections of the ensemble are assigned this slow start threshold fair share as their new slow start threshold.

Smoothed round trip time: The EFCM controller uses the current value of an aggregated smoothed round trip time of the senders of existing TCP connections of an ensemble as the initial smoothed round trip time for the sender of a new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial smoothed round trip time of the sender of the new TCP connection is set to the standard value.

Round trip time variance: The EFCM controller uses the current value of an aggregated round trip time variance of the senders of existing TCP connections of an ensemble as the initial round trip time variance for the sender of a new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial round trip time variance of the sender of the new TCP connection is set to the standard value.

The EFCM controller uses the same algorithms for one-time network-information sharing than the FS-TCBI controller. Thus, the FS-TCBI algorithms for calculating new values of some TCP control variables are reused in the EFCM controller.

5.3.6.2 EFCM Common Congestion Control for concurrent TCP Connections

Whenever standard TCP would change the value of one of the jointly controlled TCP control variables in the sender of a TCP connection, EFCM uses this change to trigger updates to these variables for all other connections within the same ensemble, according to the following rules (see Figure 5.1):

Congestion window: After every change of the congestion window in the sender of one of the existing TCP connections in an ensemble, the aggregated congestion window for this ensemble is updated by adding all current congestion windows of the senders of TCP connections in this ensemble. The aggregated value is used to calculate a fair share of congestion window which is the new congestion window for each sender of a TCP connection in an ensemble.

Slow start threshold: After every change of the slow start threshold in the sender of one of the existing TCP connections in an ensemble the aggregated slow start threshold for this ensemble is updated by adding all current slow start thresholds of the senders of TCP connections in this ensemble. The aggregated value is used to calculate a fair share of slow start threshold which is the new slow start threshold of for each sender of a TCP connection in an ensemble.

Smoothed round trip time: After every change of the smoothed round trip time in the sender of one of the n TCP connections in an ensemble the aggregated smoothed round trip time of this ensemble is updated by a weighted calculation of $(n-1)/n$ times the last value of the aggregated smoothed round trip time plus $1/n$ times the new smoothed round trip time. The senders of all TCP connections in the ensemble get this calculation result of the aggregated smoothed round trip time as their new smoothed round trip time.

Round trip time variance: After every change of the round trip time variance in the sender of one of the n TCP connections in an ensemble the aggregated round trip time variance of this ensemble is updated by a weighted calculation of $(n-1)/n$ times the last value of the aggregated round trip time variance plus $1/n$ times the new round trip time variance. The senders of all TCP connections in the ensemble get this calculation result of the aggregated round trip variance as their new round trip time variance.

It has been found out by test simulations that a non-weighted calculation of the aggregated smoothed round trip time and the aggregated round trip time variance yields in decreased performance gains of the EFCM controller. What is unknown so far is whether the selected weights based on the number of jointly controlled TCP connections in an ensemble are nearly optimal or should be replaced by other (fixed) values.

The following Figure 5.2 schematically shows how the jointly controlled TCP variables in a single ensemble are updated (steps 1. to 4.) after an ensemble event has been occurred.

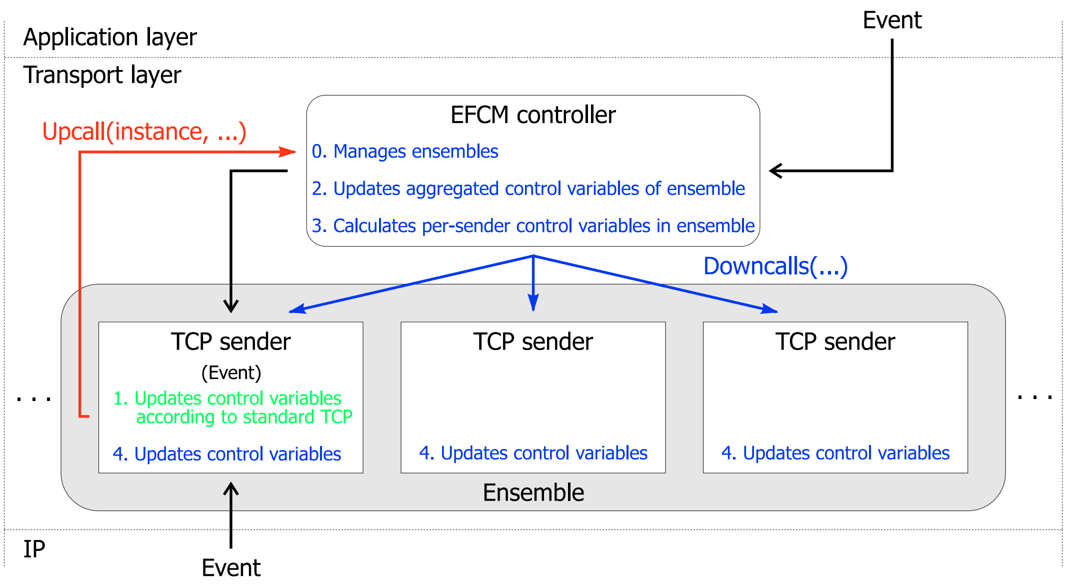

Event
Application layer
Transport layer
EFCM controller
Upcall(instance, ...)
0. Manages ensembles
2. Updates aggregated control variables of ensemble
3. Calculates per-sender control variables in ensemble
Downcalls(...)
TCP sender
(Event)
1. Updates control variables according to standard TCP
4. Updates control variables
TCP sender
4. Updates control variables
TCP sender
4. Updates control variables
. . .
. . .
Ensemble
IP
Event

Figure 5.2: Schematic description of the stepwise TCP control-variable updates in the EFCM controller

If the sender of one TCP connection in an ensemble is affected by a packet loss, the senders of all TCP connections of the ensemble will fairly reduce their congestion window and slow start threshold to the new calculated values. In Chapter 6, it will be shown that these congestion-control algorithms maintain the congestion-control paradigm of standard TCP.

If a TCP connection leaves the ensemble, i.e., the TCP connection has been recently closed, the congestion window and slow start threshold of its sender is redistributed in the ensemble: the current aggregated congestion window and the current aggregated slow start threshold are fairly shared among the senders of the remaining TCP connections in the ensemble. It might be possible but not yet considered to be necessary to limit the fair share of the aggregated slow start threshold for the senders of the remaining TCP connections in an ensemble after a TCP connection has left this ensemble. A reasonable value for this limit might be the initial slow start threshold.

5.3.6.3 EFCM Pacing Mechanism

In order to avoid a bursty sending behavior of the sender of an EFCM-controlled TCP connection, a pacing mechanism is implemented in the EFCM controller. This pacing mechanism works as follows (cf. Section 5.2.5.2): The sender of every TCP connection in an ensemble can send at most two TCP segments in a burst. The time Δt between two consecutive segment bursts of the sender of a TCP connection is calculated by using the aggregated smoothed round trip time and the aggregated congestion window of an ensemble:

$$\Delta t = \alpha \cdot \text{SRTT\_AGG}(T_k)/\text{CWND\_AGG}(T_k)$$

In the current version of the EFCM controller the pacing factor α is set to the fixed value 2. If for a single TCP sender a pacing timer with an expiration time Δt is started, it is not updated by interim changed values of the aggregated smoothed round trip time and the aggregated congestion window of its ensemble. Thus, in some cases where the aggregated congestion window tends to increase often during the pacing time Δt, e.g., this is more likely to happen for ensembles with lower round trip times, the pacing algorithm might be too conservative in sending new TCP segments compared to standard TCP. In future versions of EFCM, this pacing mechanism could be improved if the pacing factor α is adapted dependent on the current aggregated smoothed round trip time, e.g., $\alpha = f(\text{SRTT\_AGG}(t))$ using a monotonic increasing function f with a range starting from 1 for lower smoothed round trip times and growing to 2 for larger smoothed round trip times.

The remarks about the pacing mechanism of the FS-TCBI controller (cf. Section 5.2.5.2) are also valid for the pacing mechanism used in the EFCM controller.

Evidently, the EFCM computations impose some overhead in time and space. A rough estimate of the additional time and space complexity of the EFCM controller compared to standard TCP is given in the following Section 5.3.7.

5.3.7 Complexity of the EFCM Controller

The complexity of a controller can be expressed using the O-calculus [72, 73] that gives an asymptotic upper bound of a complexity metric, e.g., the time or space consumption of an algorithm or a controller. Here, the time and space complexity of the EFCM controller is compared to standard TCP by using an implementation of the EFCM controller which is optimized for time consumption (cf. Section 5.2.6).

Let v be the number of jointly controlled TCP control variables of the EFCM controller. For v_{fs} of these TCP control variables a fair share calculation, e.g., for the congestion window or the slow start threshold, and for the remaining v_{w} TCP control variables a weighted calculation, e.g., for the smoothed round trip time and the round trip time variance, is used. Let n be the number of concurrent

TCP connections which form an ensemble and are jointly controlled by the EFCM controller. For every change of one of the jointly controlled TCP control variables of the EFCM controller, the additional time consumption of the EFCM controller compared to standard TCP can be estimated as:

$$\begin{array}{lcl} \text{\# Additions } (+,-) & = & O(1) \\ \text{\# Multiplications } (\cdot,/) & = & O(1) \\ \text{\# Assignments} & = & O(n) \end{array}$$

The considered implementation of the EFCM controller has an additional space consumption compared to standard TCP whose upper bound is in $O(n \cdot v_{\text{fs}} + v_{\text{w}})$ per ensemble. If an EFCM controller has to manage e ensembles at the same time, the upper bound of the additional space consumption is in $O(e + \sum_{i=1}^{e} n_i \cdot v_{\text{fs},i} + v_{\text{w},i})$; the additional e in the equation represents the space consumption of the search list that is used to determine whether a new TCP connection can join an existing ensemble or not. The overall additional time and space complexity of the EFCM controller compared to standard TCP is low. And this additional complexity, except for the search list, is only needed if the EFCM controller has to manage TCP connections in an ensemble.

5.4 Summary

In the previous sections, the main ideas of two new common congestion control and/or one-time network-information sharing approaches are described. The following Table 5.1 summarizes the functionalities and properties of these two new approaches.

Table 5.1: Overview of the two new network-information sharing approaches

	FS-TCBI	EFCM
Ensemble-decision criterion	IP addresses	
Ensemble-creation	static	
performs ensemble / temporal one-time network-information sharing	yes / no	
performs common congestion control	no	yes
supports TCP / UDP	yes / no	
is located only in a sending end system	yes	
requires additional network traffic (e.g., probe packets)	no	
is transparent for applications	yes	
complexity of the implementation	low	medium
fairness gain of concurrent TCP connections (expected[1])	+	++
throughput gain of a new TCP connection (expected[1])	+	++
throughput gain of concurrent TCP connections (expected[1])	+	++

[1]Compared to standard TCP

It can be seen that EFCM has large advantages over FS-TCBI comparing the expected performance gains of all metrics.

5.5 Practicability of Network-Information Sharing in Internet End Systems

The FS-TCBI controller, the EFCM controller, and most other network-information sharing controllers located in Internet end systems use the IP addresses of the receiving end systems to determine whether a new TCP connection belongs to a given ensemble or not. It has been already mentioned in Section 3.3.4.1 that this can cause some problems, for example, if mechanisms like NAT or Mobile IP are used which violate the necessary address-location relation of the IP addresses. Therefore, it might be sometimes difficult or even impossible to find out which TCP connections of an end system can form an ensemble and can benefit from a one-time network-information sharing or a common congestion control. There exist some mechanisms based on measurements to find out which TCP connections share the same path or bottleneck link, for example, the mechanism explained in [59]. But these mechanisms require some time to identify those TCP connections of an end system which traverse the same bottleneck link in the network. Another solution to create an ensemble of TCP connections is based on a combination of both types of NIS approaches. Then, feedback information from RCF-capable (bottleneck) routers can be used to decide which TCP connections of a sending end system traverse the same network path (or at least the same bottleneck link) and can form an ensemble. But further research must be done on this topic to determine the constraints and possible solutions for using network-information sharing approaches in sending end systems of the Internet. Thus, making network-information sharing a practical solution is still faced with a practical challenge.

Chapter 6

Analytical Comparison of Standard TCP and EFCM Congestion Control

6.1 Introduction

It has been shown in previous work [76] and will be later shown in Chapter 9 that joint flow and congestion control for ensembles of TCP connections using EFCM indeed improves throughput. Achieving such performance improvements over TCP for some connections can be easily done by disregarding TCP's congestion-control mechanisms, taking an inappropriately large share of the network bandwidth. A potential objection against these results is that the performance benefits were due to a more aggressive behavior of EFCM than the corresponding set of individually controlled standard TCP connections would exhibit—the intuitive meaning of "aggressiveness" is the expected number of unacknowledged packets that the senders of a set of TCP connections can have at a given point in time (a formal definition is given in Section 6.3). Increasing aggressiveness is beneficial from an end system's point of view as it can inject more packets into the network per time unit; it is detrimental from the network's point of view as overly aggressive end systems will lead to congestive collapse. Aggressiveness should therefore not be increased beyond the behavior of standard TCP. But such overly increased aggressiveness could not be observed in my simulations (cf. Chapter 9), since the performance of background traffic has not been adversely affected and the packet loss rate has been even slightly decreased by introducing EFCM. Nevertheless, a convincing theoretical argument that EFCM is no more aggressive to the network than individual TCP connections has been missing so far; this chapter fills this gap.

To substantiate this claim, a closer look at the aggressiveness of both a set of standard TCP connections and the corresponding EFCM-controlled ensemble is needed. In fact, it is easy to construct valid configurations of TCP connections such that the ensemble is indeed more aggressive. However, this is not a just comparison to make: these examples where EFCM is more aggressive than standard TCP are based on a vastly different share of available bandwidth among the TCP connections (the congestion windows of their senders are very different), but as it will be shown later in this chapter, EFCM ensures a fair bandwidth allocation between TCP connections. Hence, it is appropriate to compare EFCM against standard TCP with a fair bandwidth allocation among connections of a set, since this is also a valid and even desirable (cf. Section 6.4) configuration for TCP connections.

When performing such a comparison, one has to avoid to compare against an inferior case. To exclude that possibility, the number of packets are investigated that the senders of a set of TCP connections can put onto the network after a burst of packet loss events, dependent on how the available bandwidth is initially shared among these connections. As will turn out in Section 6.4, a *fair* allocation of bandwidth is actually optimal for the expected performance of a set of TCP connections. Therefore, from an end system's point of view, EFCM ensures a behavior that would be optimal for a set of TCP connections but that standard TCP itself cannot guarantee; from the network's point of view, the remaining question is whether EFCM is about as aggressive as such a fairly sharing set of TCP connections. Section 6.5 answers this question in the affirmative. The aggressiveness of EFCM is compared to the results of Section 6.4 and it is shown that EFCM is only marginally more aggressive than a fair set of standard TCP connections.

This chapter follows these three steps: showing that EFCM ensures fairness, determining the behavior of standard TCP after errors, and then putting TCP and EFCM into perspective. But first, Section 6.2 quickly summarizes the necessary background on TCP and EFCM. This and other sections are rather compact (cf. [77]); the reader is invited to a more extended exposition in [78].

Throughout this chapter, the unit for the CWND and the SSTHRESH is segments with a maximum possible segment size and not bytes. This is done to be conform with the description of the EFCM algorithms in the previous chapter. In addition, the following analytical investigations are much easier to understand if these two congestion-control variables are expressed in units of MSSs than in bytes.

6.2 Background: Standard TCP and EFCM

The comparison of standard TCP and EFCM uses the number of packets that can be put onto the network as a metric. This number is determined by the congestion window CWND and the slow start threshold SSTHRESH of the sender of a single TCP connection or the aggregated values of an ensemble, respectively.

The relevant issue is how these variables are influenced after a packet loss is detected. Section 6.5 will show why acknowledgments are not the critical issue for this chapter (details on the treatment of acknowledgments in EFCM are included in the previous chapter and in [78]).

6.2.1 Standard TCP Congestion Control

After a packet loss event, e.g., a retransmission timer timeout or dupacks, at time T_1 in the sender of TCP connection j of a set of n standard TCP connections, the basic algorithms for the congestion window and slow start threshold computation of the sender of TCP connection i work as follows [10] (cf. Section 2.3.2.6):

$$\text{CWND}_i(T_1) = \begin{cases} 1 & i = j \\ \text{CWND}_i(T_0) & i \neq j \end{cases} \tag{6.1}$$

and

$$\text{SSTHRESH}_i(T_1) = \begin{cases} \max\{\text{CWND}_i(T_0)/2, 2\} & i = j \\ \text{SSTHRESH}_i(T_0) & i \neq j \end{cases} \tag{6.2}$$

Only basic TCP mechanisms are considered; extensions like Fast Retransmit or Fast Recovery [10, 11] are not taken into account.

6.2.2 EFCM Congestion Control

In EFCM, the loss of a packet is not assigned to the sender of an individual TCP connection; rather, the consequences are fairly shared among senders of all TCP connections that belong to an ensemble. This happens by using *aggregated* congestion window CWND_AGG and slow start threshold SSTHRESH_AGG for the entire ensemble. The aggregated values are simply the sum of the individual control variables.

After any event that influences the congestion- or flow-control of the sender of a single TCP connection, its control variables are updated according to the rules of standard TCP. Then, the sum of these control variables is fairly divided onto the control variables of the senders of all TCP connections of an ensemble as their new values. For the congestion control, this means (the sender of connection j encounters an error):

$$\begin{aligned} \text{CWND\_AGG}(T_1) &= 1 + \sum_{1 \leq i \leq n, i \neq j} \text{CWND}_i(T_0) \\ \forall i : 1 \leq i \leq n : \text{CWND}_i(T_1) &= \frac{\text{CWND\_AGG}(T_1)}{n} \end{aligned} \tag{6.3}$$

and for the slow start threshold:

$$\begin{aligned} \text{SSTHRESH\_AGG}(T_1) &= \max\{\text{CWND}_j(T_0)/2, 2\} + \sum_{1 \leq i \leq n, i \neq j} \text{SSTHRESH}_i(T_0) \\ \forall i : 1 \leq i \leq n : \text{SSTHRESH}_i(T_1) &= \frac{\text{SSTHRESH\_AGG}(T_1)}{n} \end{aligned} \tag{6.4}$$

6.3 Definitions

For the later discussion, it will be handy to talk about a set of individually controlled TCP connections in a way similar to the EFCM terminology.

Definition 1 (Virtual aggregated congestion window and slow start threshold). For a set of individually controlled TCP connections of size n, the *virtual aggregated congestion window* and *virtual aggregated slow start threshold* are the sum of the individual values of the senders of these connections.

$$\text{CWND\_AGG}^{(V)} = \sum_{i=1}^{n} \text{CWND}_i, \quad \text{SSTHRESH\_AGG}^{(V)} = \sum_{i=1}^{n} \text{SSTHRESH}_i \tag{6.5}$$

The intuitive between the number of outstanding packets and the notion of aggressiveness of a congestion-control approach is captured in Definition 2.

Definition 2 (Aggressiveness). Two flow- or congestion-control mechanisms are said to have the same aggressiveness if, starting from the same aggregated congestion window and aggregated slow start threshold and experiencing the same sequence of acknowledgment arrivals or packet loss events, their aggregated congestion window and aggregated slow start threshold are still identical. If they are not identical, the difference in aggregated congestion window and aggregated slow start threshold is a measure of their difference in aggressiveness.

Remarks: For some flow- or congestion-control mechanisms only expected values for the aggregated congestion window and aggregated slow start threshold after the sequence of acknowledgment arrivals or packet loss events can be given. Other mechanisms allow to give precise values for these aggregated variables.

6.4 Behavior of a Set of Standard TCP Connections

For the behavior of a set of standard TCP connections, two results have to be established: one result includes expressions for the expected values for (aggregated) congestion window and slow start threshold after packet losses occur, the second is a consideration about how the relative behavior of these TCP connections impacts the expected value of (aggregated) congestion window and slow start threshold after a number of consecutive packet loss events occur in the senders of these TCP connections.

It is concentrated on the error case here as the comparison between TCP and EFCM after acknowledgment arrivals is simple; this is outlined in Section 6.5.

6.4.1 Expected Values of CWND_AGG$^{(V)}$ and SSTHRESH_AGG$^{(V)}$ after Packet Loss Events

The investigations of a set of n standard TCP connections are started at time T_0. After a packet loss event at time T_1 in the sender of TCP connection j of the set, its congestion window and slow start threshold are adapted as described in Equation (6.1).

In the following, it is investigated how CWND_AGG$^{(V)}$ and SSTHRESH_AGG$^{(V)}$ decrease after a burst of l consecutive packet loss events occurs. For this analysis, a combinatorial model is used that describes how the l consecutive packet loss events are distributed over the senders of n standard TCP connections of the set. For this combinatorial model, the following simplifying assumptions are made:

- all congestion windows CWND$_i(T_0)$ and slow start thresholds SSTHRESH$_i(T_0)$ of the senders belonging to the set of n TCP connections are integers,

- the loss probability p is equal and independent for all outstanding TCP segments, and

- all time-dependencies regarding the point in time when each of the outstanding TCP segments is sent and when a packet loss event will occur that is assigned to this TCP segment, e.g., a timeout of the retransmission timeout timer, are neglected.

Under these assumptions, an urn model can be used that provides probabilities for the distribution of l consecutive packet loss events among the senders of n TCP connections in a set: If CWND_AGG$^{(V)}(T_0)$ TCP segments are outstanding, the probability $p_{i,k,l}$, $0 \leq i \leq n$, $0 \leq k \leq l \leq l_{\text{max}} = \text{CWND\_AGG}^{(V)}(T_0)$, that the sender of TCP connection i is affected by k of the l consecutive packet loss events, follows the hypergeometrical distribution [79]:

$$p_{i,k,l} = \frac{\binom{\text{CWND}_i(T_0)}{k} \cdot \binom{\text{CWND\_AGG}^{(V)}(T_0) - \text{CWND}_i(T_0)}{l-k}}{\binom{\text{CWND\_AGG}^{(V)}(T_0)}{l}}. \tag{6.6}$$

The analogy for this formula is the following problem: suppose there are CWND$_i(T_0)$ red balls and CWND_AGG$^{(V)}(T_0)$ − CWND$_i(T_0)$ blue balls in an urn. What is the probability that red appears k times if l balls are drawn without replacement?
The probability that the sender of each TCP connection i of the n TCP connections in the set is affected by exactly k_i of the l consecutive packet loss events, $k_1 + \cdots + k_n = l$, is a generalized hypergeometrical distribution:

$$p(l, k_1, \ldots, k_n) = \frac{\binom{\text{CWND}_1(T_0)}{k_1} \cdot \ldots \cdot \binom{\text{CWND}_n(T_0)}{k_n}}{\binom{\text{CWND\_AGG}^{(V)}(T_0)}{l}}. \tag{6.7}$$

The analogy for this formula is the following problem: suppose there are CWND_AGG$^{(V)}(T_0)$ balls of n different colors in an urn, for color i there are CWND$_i(T_0)$ balls in the urn. What is the probability that a specific color i appears k times if balls are drawn l times without replacement?
Starting from these formulas, expected values for the aggregated congestion window and the aggregated slow start threshold after every of the l consecutive packet loss events are deduced.

After l consecutive packet loss events, the expected value of the congestion window CWND$_i(T_l)$ of the sender of TCP connection i can be computed as follows:

$$\text{E}\left[\text{CWND}_i(T_l)\right] = p_{i,0,l} \cdot \text{CWND}_i(T_0) + \sum_{j=1}^{\text{CWND}_i(T_0)} p_{i,j,l} \cdot 1 = p_{i,0,l} \cdot \text{CWND}_i(T_0) + (1 - p_{i,0,l}). \tag{6.8}$$

After l consecutive packet loss events, the expected value of the aggregated congestion window CWND_AGG$^{(V)}(T_l)$ of the senders of the set of n TCP connections can be computed as follows:

$$\begin{aligned} \text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] &= \sum_{k_1 + \ldots + k_n = l} p(l, k_1, \ldots, k_n) \cdot \left(\sum_{1 \leq i \leq n : k_i > 0} 1 + \sum_{1 \leq i \leq n : k_i = 0} \text{CWND}_i(T_0) \right) \\ &= \sum_{i=1}^{n} \text{E}\left[\text{CWND}_i(T_l)\right]. \end{aligned} \tag{6.9}$$

Equation (6.9) shows that the expected value of the aggregated congestion window after l consecutive packet loss events can be calculated in two different ways: by a sum over all possibilities of valid packet loss distributions among the senders of n TCP connections in the set or by a sum over the expected congestion windows of every sender of the n TCP connections in the set.

With the same considerations, formulas for the expected values of the slow start threshold of the sender of a single TCP connection and the aggregated slow start threshold of the senders of a set of n TCP connections can be deduced:

$$\begin{aligned}\mathrm{E}\left[\mathrm{SSTHRESH}_i(T_l)\right] \;=\;& p_{i,0,l}\cdot \mathrm{SSTHRESH}_i(T_0) + \\ & p_{i,1,l}\cdot \max\{\mathrm{CWND}_i(T_0)/2, 2\} + 2\cdot(1-p_{i,0,l}-p_{i,1,l}) \end{aligned} \tag{6.10}$$

and

$$\begin{aligned}\mathrm{E}\left[\mathrm{SSTHRESH\_AGG}^{(V)}(T_l)\right] \;=\;& \sum_{k_1+\ldots+k_n=l} p(l,k_1,\ldots,k_n)\cdot \\ & \left(\sum_{1\le i\le n:k_i>1} 2 + \sum_{1\le i\le n:k_i=1} \max\{\mathrm{CWND}_i(T_0)/2,2\} + \sum_{1\le i\le n:k_i=0} \mathrm{SSTHRESH}_i(T_0)\right) \\ =\;& \sum_{i=1}^{n} \mathrm{E}\left[\mathrm{SSTHRESH}_i(T_l)\right]. \end{aligned} \tag{6.11}$$

6.4.2 Relative Behavior of TCP Connections

These values for the expected aggregated congestion window and aggregated slow start threshold serve as a starting point for a consideration about the relative fairness of n TCP connections. Again, the expected virtual aggregated congestion window is used as a figure of merit here, which should be maximized to allow the senders of a set of TCP connections to put as many packets onto the network as the TCP congestion-control mechanisms allow.

For the senders of a set of n standard TCP connections, there is no mechanism that would ensure fairness between them: the sender of one connection could have a large amount of outstanding packets, the sender of another one only a small number. In essence, the aggregated congestion window and aggregated slow start threshold can be arbitrarily shared between the senders of this set of TCP connections. Essentially, any combination of congestion windows $\mathrm{CWND}_i(T_0)$ and slow start thresholds $\mathrm{SSTHRESH}_i(T_0)$ that meet the following equations

$$\forall i : 1 \le i \le n : 1 \le \mathrm{CWND}_i(T_0) \le \mathrm{CWND\_AGG}^{(V)}(T_0) - (n-1) \wedge$$
$$\mathrm{CWND\_AGG}^{(V)}(T_0) = \sum_{i=1}^{n} \mathrm{CWND}_i(T_0)$$

$$\forall i : 1 \le i \le n : 2 \le \mathrm{SSTHRESH}_i(T_0) \le \mathrm{SSTHRESH\_AGG}^{(V)}(T_0) - 2\cdot(n-1) \wedge$$
$$\mathrm{SSTHRESH\_AGG}^{(V)}(T_0) = \sum_{i=1}^{n} \mathrm{SSTHRESH}_i(T_0)$$

represents a legal instance of TCP behavior. But how do these values influence the behavior of a set of connections after a burst of l consecutive packet loss events occurred? More specifically: Starting from any fixed set of $\mathrm{CWND}_i(T_0)$'s and $\mathrm{SSTHRESH}_i(T_0)$'s, how does the expected value for aggregated congestion window and aggregated slow start threshold look like after l consecutive packet loss events? It is intended to show that the expected aggregated congestion window after errors is maximized (in nearly all cases) if $|\text{CWND}_i(T_0) - \text{CWND}_j(T_0)| \leq 1$ for any $1 \leq i, j \leq n$—the result does not hold in general as there are some extreme combinations of values for which this property is slightly violated; numbers are given later.

In addition, the *overall* expected aggregated congestion window is considered:

$$\begin{aligned} \text{E}\left[\text{CWND\_AGG}^{(V)}\right] &= \sum_{l=0}^{l_{\text{max}}} \Pr[X = l] \cdot \text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] & (6.12) \\ &= \sum_{l=0}^{l_{\text{max}}} \text{gmp}(l+1, p) \cdot \text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right], & (6.13) \end{aligned}$$

where p is the packet loss probability in the Internet, X is the random variable representing the number of consecutive packet losses (X is geometrically distributed with parameter p, $\Pr[X = l] = \text{gmp}(l+1, p) = (1-p) \cdot p^l$ with $l = 0, 1, 2, \ldots$ (cf. [79])), and $l_{\text{max}} = \text{CWND\_AGG}^{(V)}(T_0)$ is the maximum number of outstanding and hence potentially lost packets. It will be shown that this overall expected value is maximized if $|\text{CWND}_i(T_0) - \text{CWND}_j(T_0)| \leq 1$, $\forall i, j : 1 \leq i, j \leq n$.

It will be also shown that the expected aggregated slow start threshold after a burst of l packet loss events is independent from the partition of the current aggregated slow start threshold $\text{SSTHRESH\_AGG}^{(V)}(T_0)$ if $\text{CWND}_i(T_0) = \text{CWND}_j(T_0)$, $\forall i, j : 1 \leq i, j \leq n$.

Lemma 1. *The expected aggregated congestion window after l packet loss events $E[\mathit{CWND\_AGG}^{(V)}(T_l)]$ is maximized in nearly all cases if $|\mathit{CWND}_i(T_0) - \mathit{CWND}_j(T_0)| \leq 1$. The expected overall aggregated congestion window $E[\mathit{CWND\_AGG}^{(V)}]$ is maximized in all cases if $|\mathit{CWND}_i(T_0) - \mathit{CWND}_j(T_0)| \leq 1$, $\forall i, j : 1 \leq i, j \leq n$.*

Proof. If there are n TCP connections in the set, each sender of them with a current congestion window $\text{CWND}_i(T_0)$, $1 \leq i \leq n$, and the current aggregated congestion window $\text{CWND\_AGG}^{(V)}(T_0) = \text{CWND}_1(T_0) + \ldots + \text{CWND}_n(T_0)$ of the set. What is investigated is how the expected aggregated congestion window of the senders of a set of n TCP connections is influenced by l consecutive packet loss events, $1 \leq l \leq l_{\text{max}} = \text{CWND\_AGG}^{(V)}(T_0)$, dependent on the choice of the current congestion windows, i.e., the following mathematical expression is considered:

$$\text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] = f(l, \text{CWND}_1(T_0), \ldots, \text{CWND}_n(T_0)) \qquad (6.14)$$

To reach this aim, only the senders of two TCP connections of the whole set are considered in more detail, w.l.o.g., the senders of TCP connection 1 and TCP connection 2. These two TCP senders have an aggregated congestion window

$$\text{CWND\_AGG}^{*(V)}(T_0) = \text{CWND}_1(T_0) + \text{CWND}_2(T_0) \qquad (6.15)$$

and can be affected by l consecutive packet loss events, with $1 \leq l \leq l^*_{\text{max}} = \text{CWND\_AGG}^{*(V)}(T_0)$. It is investigated how their expected aggregated congestion window

$$\text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right] = f(l, \text{CWND}_1(T_0), \text{CWND}_2(T_0)) \qquad (6.16)$$

after l consecutive packet loss events depends on the choice of the current congestion windows $\text{CWND}_1(T_0)$ and $\text{CWND}_2(T_0)$.

The probability $p_{k,l}$ that k of the l packet loss events occur in the sender of TCP connection 1 and $l-k$ of the l packet loss events occur in the sender of TCP connection 2, is:

$$p_{k,l} = \frac{\binom{\text{CWND}_1(T_0)}{k} \cdot \binom{\text{CWND}_2(T_0)}{l-k}}{\binom{\text{CWND}_1(T_0)+\text{CWND}_2(T_0)}{l}} \tag{6.17}$$

Let T_l be the point in time where the l-th consecutive packet loss event occurs. The expected aggregated congestion window $\text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right]$ after l packet loss events, is:

$$\begin{aligned} \text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right] &= p_{l,l} \cdot (1 + \text{CWND}_2(T_0)) + \\ &\quad p_{0,l} \cdot (1 + \text{CWND}_1(T_0)) + 2 \cdot (1 - p_{l,l} - p_{0,l}) \end{aligned} \tag{6.18}$$

In the following, the current aggregated congestion window $\text{CWND\_AGG}^{*(V)}(T_0)$ is fixed and it is investigated how the expected aggregated congestion window $\text{E}[\text{CWND\_AGG}^{*(V)}(T_l)]$ changes under different partitions of $\text{CWND\_AGG}^{*(V)}(T_0)$, i.e., by considering all possible cases

$$\begin{aligned} 1 \leq \text{CWND}_1(T_0), \text{CWND}_2(T_0) \leq \text{CWND\_AGG}^{*(V)}(T_0) - 1 : \\ \text{CWND}_1(T_0) + \text{CWND}_2(T_0) = \text{CWND\_AGG}^{*(V)}(T_0) \end{aligned} \tag{6.19}$$

Numerical calculations of these partitions (cf. Appendix in [78]) show that in nearly all cases the expected aggregated congestion window $\text{E}[\text{CWND\_AGG}^{*(V)}(T_l)]$ after l consecutive packet loss events reaches its maximum, if $\text{CWND}_1(T_0) = \text{CWND}_2(T_0)$ for even $\text{CWND\_AGG}^{*(V)}(T_0)$ or $|\text{CWND}_1(T_0) - \text{CWND}_2(T_0)| = 1$ for odd $\text{CWND\_AGG}^{*(V)}(T_0)$. There exist a few other partitions of the current aggregated congestion window which have a slightly larger expected aggregated congestion window after a specific number of consecutive packet loss events. For example, if I have a current aggregated congestion window of 32 and consider two partitions $P_1 = (\text{CWND}_1(T_0) = 16, \text{CWND}_2(T_0) = 16)$ and $P_2 = (2, 30)$, the expected aggregated congestion window after 17 consecutive packet loss events is exactly 2 for partition P_1 and ≈ 2.2117 as the value for partition P_2. But

$$\text{E}\left[\text{CWND\_AGG}^{*(V)}\right] = \sum_{l=0}^{l^*_{\text{max}}} \text{gmp}(l+1, p) \cdot \text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right], \tag{6.20}$$

with $l^*_{\text{max}} = \text{CWND\_AGG}^{*(V)}(T_0)$, is maximized only if $\text{CWND}_1(T_0) = \text{CWND}_2(T_0)$ for even $\text{CWND\_AGG}^{*(V)}(T_0)$ or if $|\text{CWND}_1(T_0) - \text{CWND}_2(T_0)| = 1$ for odd $\text{CWND\_AGG}^{*(V)}(T_0)$. For the example above and an exemplarily chosen packet loss probability of $p = 10^{-4}$ in the Internet, the result is ≈ 31.9985 for partition P_1 as the maximum possible value and ≈ 31.9973 for partition P_2 as the third smallest possible value. Of course, in this consideration, the expected value is dominated by the value for the error-free case as the packet loss probability is here rather small.

Therefore, to reach the maximum expected overall aggregated congestion window the current aggregated congestion window of the senders of the two TCP connections has to be fairly partitioned, e.g., $\text{CWND}_1(T_0) = \text{CWND}_2(T_0)$ for even $\text{CWND\_AGG}^{*(V)}(T_0)$. Since the two TCP connections are chosen w.l.o.g., this result is valid for any pair of two TCP connections of the set. Hence, the expected overall aggregated congestion window $\text{E}[\text{CWND\_AGG}^{(V)}]$ reaches its maximum only for $\text{CWND}_1(T_0) = \ldots = \text{CWND}_n(T_0)$ if $\text{CWND\_AGG}^{(V)}(T_0)$ is divisible by n without remainder or for $\forall i,j : 1 \leq i,j \leq n : |\text{CWND}_i(T_0) - \text{CWND}_j(T_0)| \leq 1$ in all other cases (if this were not the case, there are at least senders of two TCP connections which differ in congestion window and for which it has been shown that redistributing congestion window increases the expected aggregated congestion window). $\square$

Lemma 2. *In the special case $\text{CWND}_i(T_0) = \text{CWND}_j(T_0) = \text{CWND}(T_0)$, $\forall i,j : 1 \leq i,j \leq n$, the expected value of the aggregated slow start threshold after l consecutive packet loss events is independent of the partition of the current aggregated slow start threshold $\text{SSTHRESH\_AGG}^{(V)}(T_0)$.*

Proof. In the case of a fairly shared current aggregated congestion window, the probability that the sender of TCP connection i is affected by k of the l packet loss events is equal to the probability that the sender of TCP connection j is affected by k of the l packet loss events, i.e. (cf. Equation (6.6)):

$$\forall i,j : 1 \leq i,j \leq n : p_{i,k,l} = p_{j,k,l} = p_{k,l} \tag{6.21}$$

It follows (cf. Equations (6.10) and (6.11)):

$$\begin{aligned}
&\text{E}\left[\text{SSTHRESH\_AGG}^{(V)}(T_l)\right] = \sum_{i=1}^{n} \text{E}\left[\text{SSTHRESH}_i(T_l)\right] = \\
&\sum_{i=1}^{n} \left(p_{0,l} \cdot \text{SSTHRESH}_i(T_0) + p_{1,l} \cdot \max\left\{\text{CWND}(T_0)/2, 2\right\} + 2 \cdot (1 - p_{0,l} - p_{1,l})\right) = \\
&p_{0,l} \cdot \sum_{i=1}^{n} \text{SSTHRESH}_i(T_0) + p_{1,l} \cdot n \cdot \max\left\{\text{CWND}(T_0)/2, 2\right\} + 2 \cdot n \cdot (1 - p_{0,l} - p_{1,l}) = \\
&p_{0,l} \cdot \text{SSTHRESH\_AGG}^{(V)}(T_0) + p_{1,l} \cdot \max\left\{\text{CWND\_AGG}^{(V)}(T_0)/2, 2 \cdot n\right\} + \\
&2 \cdot n \cdot (1 - p_{0,l} - p_{1,l})
\end{aligned} \tag{6.22}$$

Since Equation (6.22) is independent of the partition of the current aggregated slow start threshold $\text{SSTHRESH\_AGG}^{(V)}(T_0)$, the lemma holds. $\square$

From the end system's point of view, the special case, i.e., the case where all congestion windows and slow start thresholds of the senders of a set of TCP connections are equal, is the best case for the senders of a set of n standard TCP connections regarding the expected overall aggregated congestion window as it allows the largest load to be generated after packet losses. The special case is also the expected case for senders of a set of n standard TCP connections regarding the aggregated slow start threshold. From the network's point of view, the special case represents one possible standard-conform congestion-control behavior of n separately controlled TCP connections of an end system.

6.5 EFCM Behavior and Comparison with Standard TCP

Finally, it is intended to show that EFCM only marginally differs in aggressiveness from a fairly sharing set of standard TCP connections.

6.5.1 Aggressiveness after Acknowledgment Arrival

Looking at the description of the EFCM control algorithms from the previous chapter, it is clear that after an acknowledgment arrival, standard TCP and EFCM perform the same operations on the values of congestion window and slowstart threshold for the sender of a single connection. The aggregated congestion window of the senders of n EFCM-controlled and the senders of n standard TCP connections is increased by at most one, dependent on the current aggregated slow start threshold of the senders of an ensemble or the slow start threshold of the sender of that TCP connection in the set which receives this TCP acknowledgment. This also holds if a number of acknowledgments arrive in a row.

Hence, after an acknowledgment for a so-far unacknowledged TCP segment has been received, the EFCM controller is as aggressive to the network as standard TCP.

6.5.2 Aggressiveness after Packet Loss Event

The investigation of an ensemble of n EFCM-controlled TCP connections is started at time T_0. The variables CWND$_i(T_l)$ and SSTHRESH$_i(T_l)$, CWND_AGG(T_l) and SSTHRESH_AGG(T_l) again indicate the values of these variables at time T_l when l consecutive packet loss events have occurred.

The values of the aggregated congestion window and slow start threshold are simply determined by Equations (6.3) and (6.4). Note that these equations give are *not* expected values, but the precise values—there is no random element as it does not matter which sender of the TCP connections in the ensemble encounters the packet loss. An additional advantage of EFCM is that an ensemble of EFCM-controlled TCP connections is much simpler and easier to analyze than a set of TCP connections. Determining the aggressiveness of an ensemble of n TCP connections after l consecutive packet loss events occur is then simply a matter of applying these formulas l times. This computation is easy to do numerically and the results can be compared to the numbers for standard TCP (details are presented in [78]).

Extensive investigations have been performed with various realistic current congestion windows between 1 and 128, with various realistic current slow start thresholds between 2 and 64, and with different numbers n of standard TCP connections in a set or in an EFCM ensemble. As an example, Table 6.1 shows the absolute difference of the aggregated congestion windows $\Delta_{n,l}^{\text{CWND}} = \text{CWND\_AGG}^{(V)}(T_l) - \text{CWND\_AGG}(T_l)$, and Table 6.2 shows the absolute difference of the aggregated slow start thresholds $\Delta_{n,l}^{\text{SSTHRESH}} = \text{SSTHRESH\_AGG}^{(V)}(T_l) - \text{SSTHRESH\_AGG}(T_l)$ for a current congestion window of CWND$_i(T_0) = 32$ and a current slow start threshold of SSTHRESH$_i(T_0) = 64$, $1 \leq i \leq n$. For other current congestion windows and for other current slow start thresholds the results are comparable to the stated ones.

The overall result is that the senders of a set of n standard TCP connections will have an expected aggregated congestion window equal or slightly lower than the aggregated congestion window of the

Table 6.1: Absolute difference $\Delta^{\text{CWND}}_{n,l}$ of aggregated congestion windows

l	$\Delta^{\text{CWND}}_{1,l}$	$\Delta^{\text{CWND}}_{2,l}$	$\Delta^{\text{CWND}}_{3,l}$	$\Delta^{\text{CWND}}_{4,l}$	$\Delta^{\text{CWND}}_{5,l}$	$\Delta^{\text{CWND}}_{10,l}$	$\Delta^{\text{CWND}}_{20,l}$	$\Delta^{\text{CWND}}_{50,l}$
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	– 0.24603	– 0.21754	– 0.18307	– 0.15597	– 0.08746	– 0.04609	– 0.01900
3	0.00000	– 0.36905	– 0.43663	– 0.41336	– 0.37552	– 0.23658	– 0.13148	– 0.05588
4	0.00000	– 0.36602	– 0.58269	– 0.62142	– 0.60227	– 0.42657	– 0.25004	– 0.10957
5	0.00000	– 0.29998	– 0.64625	– 0.77746	– 0.80431	– 0.64083	– 0.39627	– 0.17904
6	0.00000	– 0.21938	– 0.64330	– 0.87423	– 0.96594	– 0.86629	– 0.56518	– 0.26329
7	0.00000	– 0.14845	– 0.59601	– 0.91627	– 1.08183	– 1.09281	– 0.75231	– 0.36137
8	0.00000	– 0.09485	– 0.52441	– 0.91334	– 1.15301	– 1.31268	– 0.95368	– 0.47237
9	0.00000	– 0.05795	– 0.44368	– 0.87668	– 1.18401	– 1.52021	– 1.16572	– 0.59540
10	0.00000	– 0.03414	– 0.36391	– 0.81698	– 1.18108	– 1.71135	– 1.38527	– 0.72963
15	0.00000	– 0.00167	– 0.09897	– 0.42975	– 0.88078	– 2.35926	– 2.50840	– 1.54157
20	0.00000	– 0.00006	– 0.01953	– 0.16920	– 0.49864	– 2.51091	– 3.51882	– 2.52548
25	0.00000	0.00000	– 0.00319	– 0.05618	– 0.24114	– 2.32104	– 4.30274	– 3.60958
30	0.00000	0.00000	– 0.00046	– 0.01662	– 0.10491	– 1.96110	– 4.82650	– 4.73699
35	0.00000	0.00000	– 0.00006	– 0.00452	– 0.04224	– 1.55569	– 5.10169	– 5.86325
40	0.00000	0.00000	– 0.00001	– 0.00116	– 0.01603	– 1.17715	– 5.16267	– 6.95432
45	0.00000	0.00000	0.00000	– 0.00029	– 0.00581	– 0.85831	– 5.05270	– 7.98480
50	0.00000	0.00000	0.00000	– 0.00007	– 0.00203	– 0.60726	– 4.81564	– 8.93642
60	0.00000	0.00000	0.00000	0.00000	– 0.00023	– 0.28293	– 4.11436	–10.55809
70	0.00000	0.00000	0.00000	0.00000	– 0.00002	– 0.12234	– 3.30455	–11.77141
80	0.00000	0.00000	0.00000	0.00000	0.00000	– 0.04994	– 2.53393	–12.57794
90	0.00000	0.00000	0.00000	0.00000	0.00000	– 0.01949	– 1.87357	–13.00907
100	0.00000	0.00000	0.00000	0.00000	0.00000	– 0.00734	– 1.34491	–13.11227

senders of n EFCM-controlled TCP connections. More precisely, for realistic numbers of jointly controlled TCP connections the congestion window of the sender of a single EFCM-controlled TCP connection is at most 0.30 (segments) larger than the congestion window of the sender of a single standard TCP connection. In contrast, the senders of a set of n standard TCP connections will have an aggregated slow start threshold that is in most cases equal to or (slightly) larger than the expected aggregated slow start threshold of the senders of n EFCM-controlled TCP connections. The differences in the aggregated congestion windows are much lower than the differences in the aggregated slow start thresholds. Therefore, the senders of a set of n standard TCP connections are slightly more conservative than the senders of an ensemble of n EFCM-controlled TCP connections in sending new TCP segments after a burst of packet loss events, but they are (slightly) more aggressive in sending new TCP segments after the reception of a

Table 6.2: Absolute difference $\Delta_{n,l}^{\text{SSTHRESH}}$ of aggregated slow start thresholds

l	$\Delta_{1,l}^{\text{SSTHRESH}}$	$\Delta_{2,l}^{\text{SSTHRESH}}$	$\Delta_{3,l}^{\text{SSTHRESH}}$	$\Delta_{4,l}^{\text{SSTHRESH}}$	$\Delta_{5,l}^{\text{SSTHRESH}}$	$\Delta_{10,l}^{\text{SSTHRESH}}$	$\Delta_{20,l}^{\text{SSTHRESH}}$	$\Delta_{50,l}^{\text{SSTHRESH}}$
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	+ 0.48016	+ 0.26140	+ 0.17421	+ 0.12893	+ 0.05408	+ 0.02445	+ 0.00916
3	0.00000	+ 0.92857	+ 0.59023	+ 0.42278	+ 0.32641	+ 0.14887	+ 0.07023	+ 0.02701
4	0.00000	+ 1.32499	+ 0.93180	+ 0.70332	+ 0.56107	+ 0.27456	+ 0.13465	+ 0.05308
5	0.00000	+ 1.15716	+ 1.27695	+ 1.00056	+ 0.81800	+ 0.42405	+ 0.21541	+ 0.08696
6	0.00000	+ 0.73347	+ 1.62516	+ 1.31015	+ 1.09089	+ 0.59231	+ 0.31054	+ 0.12825
7	0.00000	+ 0.39295	+ 1.83406	+ 1.63067	+ 1.37735	+ 0.77589	+ 0.41838	+ 0.17657
8	0.00000	+ 0.18306	+ 1.62896	+ 1.96024	+ 1.67624	+ 0.97245	+ 0.53751	+ 0.23156
9	0.00000	+ 0.07200	+ 1.28838	+ 2.29561	+ 1.98641	+ 1.18047	+ 0.66676	+ 0.29290
10	0.00000	+ 0.02048	+ 0.94535	+ 2.29606	+ 2.30606	+ 1.39901	+ 0.80513	+ 0.36026
15	0.00000	− 0.00311	+ 0.08193	+ 0.88710	+ 2.24321	+ 2.63398	+ 1.61011	+ 0.77799
20	0.00000	− 0.00018	− 0.01273	+ 0.14707	+ 0.88211	+ 4.07566	+ 2.56879	+ 1.30991
25	0.00000	− 0.00001	− 0.00557	− 0.01040	+ 0.21060	+ 5.27421	+ 3.66120	+ 1.93468
30	0.00000	0.00000	− 0.00114	− 0.01752	+ 0.00600	+ 4.53959	+ 4.87954	+ 2.63769
35	0.00000	0.00000	− 0.00018	− 0.00788	− 0.02754	+ 3.18892	+ 6.21442	+ 3.40921
40	0.00000	0.00000	− 0.00002	− 0.00261	− 0.02009	+ 1.97537	+ 7.65013	+ 4.24308
45	0.00000	0.00000	0.00000	− 0.00074	− 0.01015	+ 1.09606	+ 9.16449	+ 5.13551
50	0.00000	0.00000	0.00000	− 0.00019	− 0.00435	+ 0.53331	+10.18760	+ 6.08430
60	0.00000	0.00000	0.00000	− 0.00001	− 0.00061	+ 0.03748	+ 9.05773	+ 8.14613
70	0.00000	0.00000	0.00000	0.00000	− 0.00007	− 0.06409	+ 6.51607	+10.42121
80	0.00000	0.00000	0.00000	0.00000	− 0.00001	− 0.05433	+ 4.12677	+12.90010
90	0.00000	0.00000	0.00000	0.00000	0.00000	− 0.03042	+ 2.34378	+15.56695
100	0.00000	0.00000	0.00000	0.00000	0.00000	− 0.01428	+ 1.17269	+18.39861

new TCP acknowledgment.

Only in some academic cases where the current congestion windows are very low and the current slow start thresholds are very high, e.g., all n TCP connections of the set or ensemble are recently started, the aggregated slow start threshold of the senders of n EFCM-controlled TCP connections is (much) less reduced than the aggregated slow start threshold of the senders of n standard TCP connections. But since the congestion windows of the senders of n EFCM-controlled TCP connections are decreased comparably as the congestion windows of the senders of n standard TCP connections even in these cases, the overall possible number of outstanding TCP segments after the packet loss event burst is similar until new TCP acknowledgments arrive. Then, the senders of active EFCM-controlled TCP connections will increase their aggregated congestion window (much) faster than the senders of active standard TCP connections, but much lower than the senders of new standard TCP connections will do. In the end, the EFCM controller does not violate any congestion-control paradigms of standard TCP.

6.6 Conclusion and Outlook

This chapter has established three main results: First, it has been shown that the EFCM approach results in a fair sharing of bandwidth among an ensemble of jointly controlled TCP connections. Second, the aggressiveness of a set of standard TCP connections has been considered and it has been shown that aggressiveness of such a set is (stochastically) optimal when all standard TCP connections equally share in the available bandwidth; then, the end system can put the highest amount of packets onto the network but still behaves within the limitations of TCP's congestion-control mechanism. Third, an EFCM ensemble has been compared against the corresponding set of fairly sharing TCP connections and it has been shown that the aggressiveness of these two approaches is largely identically.

These results together can justify the appropriateness of EFCM for Internet usage and mollify potential objections against its usage of available bandwidth and potential harming of background traffic. Moreover, they also allow a new interpretation of the performance benefits of EFCM. While EFCM does provide some performance enhancements by mechanisms that were not discussed in this chapter (e.g., a more appropriate setting of round trip timers), it seems that its fairness properties are to a large extent responsible for the performance improvements. The reason is that with an unfair sharing of bandwidth, TCP connections with a large share of bandwidth have a larger probability of encountering a packet loss, drastically reducing the total number of packets the end system can send. In a stochastic sense, TCP connections which consume a large share of the available bandwidth are bad for the entire set. By ensuring fair resource sharing, this effect is reduced and EFCM cushions the expected performance of a set of TCP connections.

Earlier mathematical models of TCP, e.g., those developed and evaluated in [52], [53], and [80], are mainly focused on approximations of the long-term throughput a single TCP connection receives over a network path with a given (mean) round trip time and (mean) packet-loss probability. In contrast, the mathematical model of TCP described in this chapter analyses the behavior of a set of TCP connections of a single end system regarding the aggressiveness of this set to the network dependent on initial control-variable settings, packet-loss and acknowledgment-arrival events. Thus, with this new TCP model a better understanding of the relation between network events and (maximum) aggressiveness of TCP connections of a single end system to the network has been reached. This might be beneficial for other mathematical investigations of TCP-based traffic.

Chapter 7

Network-Information Sharing in Internet End Systems: Performance-Evaluation Overview

7.1 Introduction

This chapter describes and explains the calculations of the performance metrics used for the performance evaluation of standard TCP compared to the NIS approaches FS-TCBI and EFCM in an Internet end system. In addition, the simulation model used for the performance evaluation of standard TCP and the NIS approaches FS-TCBI and EFCM is depicted in detail.

7.2 Performance-Evaluation Metrics

The main performance metrics used for the performance evaluation of standard TCP compared to FS-TCBI and EFCM are the throughput of and the fairness between TCP connections of an Internet end system. For the calculation of these performance metrics only some of the TCP connections of an end system are considered. These TCP connections are—at least potentially—able to share network information, since useful network information is available at the end system. In other words, if the NIS approach is enabled in the end system, these TCP connections are influenced by the congestion-control algorithms of FS-TCBI or EFCM. Therefore, these TCP connections are called FS-TCBI- or EFCM-controllable TCP connections (both terms can be used as synonyms).

In general, the percentage of EFCM-controllable TCP connections depends on the type of the sending end system. For example, in a large WWW or proxy server with numerous TCP connections the probability of using FS-TCBI or EFCM will be (much) higher compared to an ordinary sending end system with its few TCP connections. In order not to reflect this sending end-system dependency in the evaluation, the metric computations were restricted to the EFCM-controllable TCP connections and are therefore independent of the type of the sending end system. The overall performance of FS-TCBI or EFCM on the throughput of all TCP connections of a sending end system can then be approximated by considering the percentage of EFCM-controllable TCP connections on all TCP connections of the sending end system.

In the following, all performance metrics used to evaluate the performance of standard TCP compared to FS-TCBI and EFCM are described.

7.2.1 Throughput

The two mean throughput metrics defined in Section 2.3.2.7 are used to compare the overall throughput performance of EFCM-controllable TCP connections either controlled by standard TCP or by one of the NIS approaches. The throughputs of new EFCM-controllable TCP connections entering an ensemble are measured over their whole lifetimes d_i. For concurrent EFCM-controllable TCP connections these measurements are only performed during their times of concurrency c_i (see Figure 7.1). Another difference

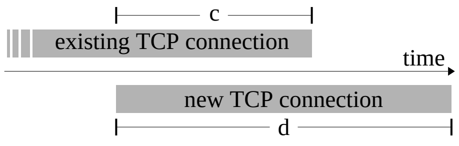


Figure 7.1: Time periods considered for the throughput calculations

between these two throughput measurements is also the type of segments which are considered during the measurement intervals. The throughputs of *new* EFCM-controllable TCP connections entering an ensemble are measured by considering all sent and acknowledged segments containing payload data, i.e., the SYN segments and their acknowledgments SYN/ACK are not counted. For *existing concurrent*

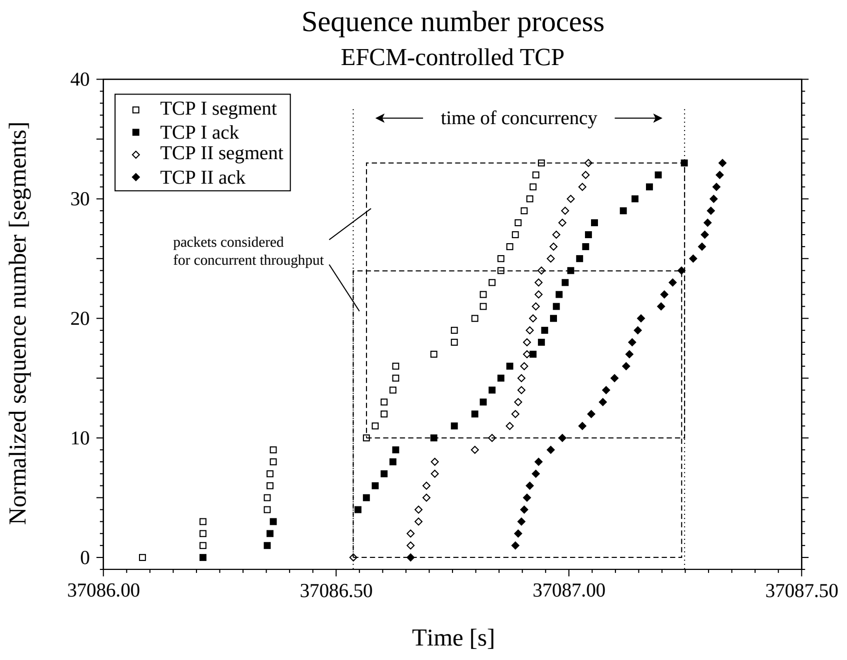


Figure 7.2: Segments considered for the throughput calculation of concurrent TCP connections

EFCM-controllable TCP connections of an ensemble the same count of segments is used. But for *new concurrent* EFCM-controllable TCP connections entering an ensemble all sent and acknowledged segments including the SYN segments and their acknowledgments SYN/ACK are counted. An example is shown in Figure 7.2 (cf. Figure 9.7). The reason to count also the SYN and SYN/ACK segments in this specific case is to obtain more expressive values for the (short-term) fairness (cf. Section 7.2.2) between concurrent TCP connections whose time of concurrency is relative short, e.g., less than two round trip times. Otherwise, the throughputs of new TCP connections entering an ensemble cannot be measured in this short measurement interval that the fairness index during this measurement interval cannot be calculated.

7.2.2 Fairness

The mean fairness index defined in Section 2.3.2.7 is used to compare the fairness of concurrent EFCM-controllable TCP connections either controlled by standard TCP or controlled by FS-TCBI or EFCM.

7.2.3 Other Metrics

For all new TCP connections entering an ensemble, the mean initial congestion window, the mean initial slow start threshold, the mean initial smoothed round trip time, and the mean initial round trip time variance are considered.

These secondary metrics are mainly used to show which influences the one-time network-information sharing algorithms of FS-TCBI and EFCM have on these control variables of EFCM-controllable TCP connections. From these metrics the mean initial congestion window is the most important one, since it gives a boundary that shows which TCP connections can send all their packets in a single (paced) burst and largely benefit from the one-time network-information sharing mechanism provided by FS-TCBI and EFCM.

7.3 Simulation Model

Evaluating the throughput and (short-term) fairness gain of FS-TCBI and EFCM compared to standard TCP is done by simulations. The network topology for these simulations is intended to reflect common client-server [18] applications in which the client end systems, i.e. the receiving end systems, obtain information from applications running on the (optionally) FS-TCBI- or EFCM-controlled server end system, the (FS-TCBI- or EFCM-capable) sending end systems. In reasonable Internet setups, a server end system is connected to the Internet via an ISP's backbone network with high-bandwidth and reliable links. Typical client end systems are connected to the Internet via networks with much smaller bandwidths, e.g., DSL lines or (low-speed) LANs. These networks can be both reliable or, in case of modern wireless LAN installations, unreliable. The backbone network itself has a bandwidth-delay product varying over time dependent on its current load. The simulated network topology is chosen with regard to these reasonable assumptions about the current Internet environment.

Other factors that influence the performance of FS-TCBI or EFCM are, for example, the number of TCP senders in the (FS-TCBI- or EFCM-capable) sending end system, the type of applications running on the (FS-TCBI- or EFCM-capable) sending end system, the round trip time between the (FS-TCBI- or EFCM-capable) sending end system and the receiving end system(s), the packet loss rate in the links between the (FS-TCBI- or EFCM-capable) sending end system and the receiving end system(s), the background traffic load in the path(s) from the (FS-TCBI- or EFCM-capable) sending end system to the receiving end system(s), the TCP variant used in the end systems, and the maximum segment size (MSS) used in the TCP connections. For each of these factors a reasonable subset of values should be defined and for every combination of parameter settings the performance of standard TCP has to be compared with the performance of the FS-TCBI or EFCM approach to reach a complete performance evaluation. This cannot be done. Thus, to obtain significant performance results also with a limited performance evaluation only one specific network model but with a reasonable choice of network parameters is considered.

In this section, the specific network model used for the performance evaluation of standard TCP, FS-TCBI, and EFCM is described in detail. The network topology of this network model is described in the following Section 7.3.1. And the load models used in the network model to generate load in the network are explained in Section 7.3.2.

7.3.1 Network Topology

The investigation of the performance of FS-TCBI or EFCM compared to standard TCP is done by using a simulated network topology shown in Figure 7.3. The simulated network topology consists of five TCP

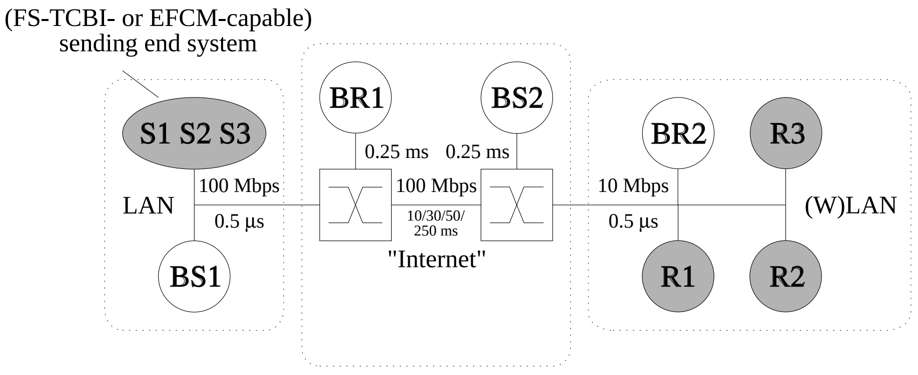


Figure 7.3: Structure of the simulated network topology

senders (S1,..., S3 and BS1, BS2) and TCP receivers (R1,..., R3 and BR1, BR2) located in several end systems, two routers, and two Ethernet-based LANs, one containing sending and systems (sender LAN) and the other containing receiving end systems (receiver LAN).

The TCP senders S1, S2, and S3 are located in a single sending end system which can be optionally equipped with a FS-TCBI or an EFCM controller. The TCP senders BS1 and BS2 are located in other sending end systems and generate background traffic. The sending end system of the background-traffic TCP sender BS1 and the FS-TCBI- or EFCM-capable sending end system are connected to the network via the sender LAN. The sending end system of the background-traffic TCP sender BS2 is connected to the network via a link with a bit rate of 100 Mbps and a propagation delay of 0.25 ms. The receiving end systems of the TCP receivers R1, R2, and R3 and the background-traffic TCP receiver BR2 are connected to the network via the receiver LAN.

The sender LAN is characterized by a bit rate of 100 Mbps and a propagation delay of 0.5 μs. The receiver LAN consists of an Ethernet-type shared medium with an overall bit rate of 10 Mbps and a propagation delay of 0.5 μs. The packet loss rate in this receiver LAN is adjustable to investigate the influence of different packet loss rates in the last hop of TCP connections on the overall throughput of these TCP connections. Hence, with this receiver LAN either a reliable (wired) Ethernet-based LAN or an unreliable wireless LAN (WLAN) with link errors can be modeled. In the scenario with a reliable last hop no errors occur in the receiver LAN. In the scenario with an unreliable last hop packets can be lost with a fixed packet loss rate of 5 %.

The routers are connected via links with a bit rate of 100 Mbps and an adjustable propagation delay, e.g., 50 ms. For each incoming link the routers have a queueing capacity of 20 IP packets. This relative low queueing capacity is chosen that packet losses in the routers due to congestion are likely to occur.

It would be particularly of interest to consider the impact of various values for the round trip time, the number of TCP senders in an FS-TCBI- or EFCM-capable sending end system, the maximum size of TCP segments, and the packet loss rate as important factors for the performance gain of FS-TCBI or EFCM.

In this dissertation, the performance evaluation of FS-TCBI and EFCM compared to standard TCP concentrates on varying the round trip time; other network parameters are kept constant using reasonable assumptions about the current Internet environment. For example, the MSS of all TCP connection considered in the simulations is set to 1460 bytes.

7.3.2 Traffi c-Load Models

Properly characterizing traffic loads for interactive Internet users is a difficult undertaking. It is decided to test the performance of the FS-TCBI and EFCM controller by considering traffic generated by a WWW traffic model [81]. This traffic model is derived from real HTTP (version 1.0) traces in corporate and educational environments and uses three abstraction levels: The session level, the page level, and the packet level. Here, a simplified version of this model is performed which consists only of the first two levels, since the packet level is determined by the underlying transport protocol used in the simulations.

In every WWW session a log-normally distributed number of WWW pages with Pareto-distributed page sizes are sent. The time between the pages, i.e., the inter-connection or reading time, is gamma distributed. The load in the network can be easily adjusted by using the same exponentially distributed session inter-arrival time, the inter-session time, with a different parameter. The distributions and parameters chosen for the stochastic variables of the simplified WWW traffic model are summarized in the following Table 7.1.

Each WWW page is transported by a single TCP connection from a sending end system (server) to a receiving end system (client). The TCP connections used to transfer consecutive pages of a session do not overlap in time, since the reading or inter-page time is started after the previous page has been

Table 7.1: Distributions and parameters for the stochastic variables of the simplified WWW model

Stochastic variable	Distribution	Distribution parameter(s)
Inter-session time	Exponential	$\mu = 5.0$ s
Pages per session (pps)	Lognormal	$\mu = 25.807$ pps $\sigma = 78.752$ pps
Inter-page time (reading time)	Gamma	$\mu = 35.286$ s $\sigma = 147.390$ s
Page size	Pareto	$\alpha = 1.7584$ $\beta = 30458$ Bytes

completely transferred.

Network traffic generated by applications working on top of TCP senders of the (optionally) FS-TCBI- or EFCM-capable sending end system follows this simplified WWW traffic model. But network traffic generated by applications working on top of background TCP senders follows a related traffic model in which the distributions of the inter-connection time and inter-session time are modified, i.e., both stochastic variables are deterministically set to zero. As a result, background-traffic TCP connections are immediately restarted once they have been terminated. This is done to reach a higher load in the network with these few background-traffic TCP senders.

7.3.3 Implementation Details

The whole simulation model is implemented in ns-2 (version 2.1b9) [61]. All TCP connections in the simulations are based on the ns-2 implementation of one-way TCP NewReno which models one direction to transfer payload between a TCP sender and a TCP receiver. The background TCP senders are instances of the ns-2 class of a TCP NewReno sender, all TCP receivers are instances of the ns-2 class of a TCP NewReno receiver. In ns-2, a TCP NewReno connection is opened by the sender by using the connection-setup mechanism of TCP (cf. Section 2.3.2.2). But a TCP NewReno connection is closed by explicitly resetting the TCP sender and receiver instance after the last segment of a TCP connection has been acknowledged at the sender without performing the connection-teardown mechanism of TCP. Thus, network information collected by a single TCP connection in a FS-TCBI- or EFCM-capable sending end system is in the worst case approximately one round trip time earlier removed in ns-2 than in a real Internet end system.

The TCP senders located in the (optionally) FS-TCBI- or EFCM-capable sending end system are instances of a new TCP sender class, called EFCM_TCP, derived from the ns-2 class of a TCP NewReno sender. This new TCP sender class provides additional network-information reuse and common congestion-control mechanisms between the TCP connections of a FS-TCBI- or EFCM-controlled ensemble. These mechanisms can be disabled or enabled to compare the performance of standard TCP connections with the performance of either FS-TCBI-controlled or EFCM-controlled TCP connections. In addition, the capability to trace control variables, to record the point in time where segments are sent and acknowledgments are received, to calculate the different performance metrics, and to evaluate these performance metrics by statistical performance-evaluation methods are also included in this new class.

In order to count packet losses in queues, to trace the queue length, and to generate histograms of the queue length, the basic link queue class of ns-2 has been adapted.

Chapter 8

Performance Evaluation of FS-TCBI by Simulations

8.1 Introduction

Simulations using the simulation model described in the previous chapter are performed with the standard TCP controller and with the FS-TCBI controller in the (FS-TCBI-capable) sending end system. The last hop in this simulation model is either reliable or has a fixed packet loss rate (PLR) of 5 % in the receiver LAN. In addition, the minimum round trip time is varied to investigate the performance of the different controllers in the sending end system dependent on this network parameter. This is done by changing the propagation delay between the two routers in the simulated network topology.

In this chapter, the simulation results for the two controllers considering several simulation scenarios with varied minimum round trip times and two last hop scenarios are shown in detail. Section 8.2 contains a general overview about the performed simulations. The following Sections 8.3 to 8.6 show the simulation results for the different simulation scenarios in detail. A summary and discussion of these simulation results can be found in Section 8.7. Finally, conclusions and an outlook to future research in the context of one-time network-information sharing approaches are given in Section 8.8.

8.2 Overview

For both last hop scenarios the different controllers (standard TCP/no FS-TCBI, FS-TCBI) are investigated for a simulated time of 250000 s in each simulation run to reach more expressive values for the mean performance metrics. This means that, for example, in the case with a minimum round trip time of 100 ms approximately 21900 TCP connections in the reliable last hop scenarios and approximately 21250 TCP connections in the unreliable last hop scenarios starting at the (FS-TCBI-capable) sending end system can be observed in the simulated time. Only some of them, i.e., those TCP connections which have concurrent TCP connections, are controlled or could be controlled by the new network-information sharing mechanisms provided by the FS-TCBI controller. In the simulation model and with the chosen traffic load model the percentage of having concurrent TCP connections is relatively low. An average computation over all simulations with a minimum round trip time of 100 ms shows that approximately 8.5 % of the new TCP connections in the reliable last hop scenarios and approximately 14.0 % of the new TCP connections in the unreliable last hop scenarios are controlled or could be controlled by the FS-TCBI approach. This amount of FS-TCBI-controllable TCP connections is proportional to the minimum round trip time between the TCP senders and the TCP receivers and increases to values up to 18.1 % for the reliable last hop and 32.2 % for the unreliable last hop in the case with a minimum round trip time of 500 ms.

In all simulations with the FS-TCBI controller and with the observed occurrence of having concurrent TCP connections the mean throughput of the background TCP connections is not negatively affected by these FS-TCBI-controlled TCP connections. Starting from these simulation results, it can be assumed that FS-TCBI-controlled TCP connections are no more aggressive to the network than standard TCP connections. This important result can be also concluded from the mean packet loss rates in the bottleneck router observed during the whole simulated time of a single simulation run and averaged over all simulation runs of a simulation scenario. The following Table 8.1 compares the mean PLRs observed in the simulation scenarios with standard TCP connections with the mean PLRs observed in the simulation scenarios with FS-TCBI-controlled TCP connections.

Table 8.1: Packet loss rates of standard TCP and FS-TCBI-controlled TCP connections

Scenario	no FS-TCBI	FS-TCBI	Difference (absolute, in %)
RTT $\geq$ 20 ms, reliable last hop	0.003263	0.003273	$-0.000010, \quad - \ 0.3055$
RTT $\geq$ 60 ms, reliable last hop	0.003257	0.003257	$+0.000000, \quad + \ 0.0000$
RTT $\geq$ 100 ms, reliable last hop	0.003275	0.003266	$+0.000009, \quad + \ 0.2756$
RTT $\geq$ 500 ms, reliable last hop	0.003225	0.003234	$-0.000009, \quad - \ 0.2783$
RTT $\geq$ 20 ms, unreliable last hop	0.000028	0.000027	$+0.000001, \quad + \ 3.7037$
RTT $\geq$ 60 ms, unreliable last hop	0.000028	0.000028	$+0.000000, \quad + \ 0.0000$
RTT $\geq$ 100 ms, unreliable last hop	0.000027	0.000028	$-0.000001, \quad - \ 3.5714$
RTT $\geq$ 500 ms, unreliable last hop	0.000027	0.000026	$+0.000001, \quad + \ 3.8462$

In the following Tables 8.2 to 8.16, TCP 1, TCP 2, and TCP 3 are FS-TCBI-controllable TCP connections of the (FS-TCBI-capable) sending end system transferring WWW data. The simulation results of each simulation scenario shown in these tables are averages over twelve independent simulation runs, each of these runs is performed for a large simulated time of 250000 seconds. The transient analysis of the simulations (cf. Appendix A) shows that the throughputs of the TCP connections in the (FS-TCBI-capable) sending end system have no transient phase. Since all considered performance metrics depend on the observed throughputs of these TCP connections, all throughput observations during the whole simulated time can be used to calculate the mean values of the different performance metrics.

Both stated mean throughput metrics ($\overline{TPO}$, $\overline{TPC}$) of the TCP connections are expressed in segments per second. For the new TCP connections entering an ensemble also the mean initial congestion window ($\overline{\text{ICWND}}$), the mean initial slow start threshold ($\overline{\text{ISSTHRESH}}$), the mean initial smoothed round trip time ($\overline{\text{ISRTT}}$), and the mean initial round trip time variance ($\overline{\text{IRTTVAR}}$) of their senders are shown. For the concurrent TCP connections of an ensemble also both mean throughput metrics ($\overline{TPO}$, $\overline{TPC}$) and the mean fairness index ($\overline{I}_f$) are shown.

The last column Δ in these tables denotes whether the simulation results are statistically significantly different or not for a given confidence level. A '+' or '-' denotes that the FS-TCBI controller or standard TCP is significantly better. A '=' means that with the simulation results no significant difference between the FS-TCBI and the standard TCP controller can be concluded. The details of the statistical evaluation of the simulation results can be found in Appendix B.2 of this dissertation.

For the senders of standard TCP connections the mean initial smoothed round trip time and the mean initial round trip time variance are not applicable (N/A) for the computation of the initial retransmission timeout timer, i.e., the initial retransmission timeout timer is set to the fixed standard value of 6 s.

8.3 Simulation 1: TCP NewReno, RTT $\geq$ 20 ms

8.3.1 Reliable Last Hop

Table 8.2 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, for new TCP connections the FS-TCBI controller is able to increase the overall mean throughput by approximately 10 % and the connection-oriented mean throughput by approximately 3 %.

Table 8.2: Simulation results of simulation 1, scenario 1 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	42.25	46.92	+
	TCP 2	42.30	45.89	+
	TCP 3	42.12	46.88	+
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	46.61	48.16	+
	TCP 2	46.74	47.93	+
	TCP 3	46.89	48.40	+
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	9.39	
	TCP 2	3.00	9.08	
	TCP 3	3.00	9.14	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.85	
	TCP 2	44.00	40.77	
	TCP 3	44.00	40.78	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.113	
	TCP 2	N/A	0.112	
	TCP 3	N/A	0.113	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.066	
	TCP 2	N/A	0.065	
	TCP 3	N/A	0.066	

Table 8.3 shows the simulation results for concurrent TCP connections of an ensemble. The overall mean throughput and the connection-oriented mean throughput of concurrent TCP connections are slightly reduced by approximately 3 % if the FS-TCBI controller is used. The mean fairness between concurrent TCP connections of an ensemble is not affected by the FS-TCBI controller.

Table 8.3: Simulation results of simulation 1, scenario 1 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	39.97	38.69	$-$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	38.52	37.42	$-$
$\overline{I}_f$ of concurrent TCP connections []	0.7916	0.7896	$=$

8.3.2 Unreliable Last Hop

Table 8.4 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, the overall mean throughput of new TCP connections is largely increased by approximately 48 % if the FS-TCBI controller is used. But the connection-oriented mean throughput is slightly decreased by approximately 2 %.

Table 8.4: Simulation results of simulation 1, scenario 2 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	23.56	35.51	$+$
	TCP 2	24.20	35.88	$+$
	TCP 3	24.99	36.62	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	69.12	68.36	$=$
	TCP 2	69.69	67.87	$-$
	TCP 3	69.78	68.24	$-$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	3.99	
	TCP 2	3.00	3.99	
	TCP 3	3.00	4.01	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	33.37	
	TCP 2	44.00	33.20	
	TCP 3	44.00	33.21	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.036	
	TCP 2	N/A	0.036	
	TCP 3	N/A	0.036	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.018	
	TCP 2	N/A	0.019	
	TCP 3	N/A	0.019	

Table 8.5 shows the simulation results for concurrent TCP connections of an ensemble. If the FS-TCBI controller is used, the overall mean throughput of these TCP connections is increased by approximately 12 % compared to standard TCP. But this comes along with an approximately 5 % decreased connection-oriented mean throughput. The mean fairness between concurrent TCP connections of an ensemble is not affected by the FS-TCBI controller.

Table 8.5: Simulation results of simulation 1, scenario 2 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	35.48	39.63	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	65.65	62.30	$-$
$\overline{I}_f$ of concurrent TCP connections []	0.8261	0.8244	$=$

8.4 Simulation 2: TCP NewReno, RTT $\geq$ 60 ms

8.4.1 Reliable Last Hop

Table 8.6 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, for new TCP connections the FS-TCBI controller is able to increase the overall mean throughput by approximately 12 % and the connection-oriented mean throughput by approximately 6 %.

Table 8.6: Simulation results of simulation 2, scenario 1 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	33.82	37.84	$+$
	TCP 2	33.98	37.59	$+$
	TCP 3	33.35	37.79	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	37.12	39.54	$+$
	TCP 2	37.49	39.35	$+$
	TCP 3	37.42	39.40	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	8.88	
	TCP 2	3.00	8.83	
	TCP 3	3.00	8.83	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.09	
	TCP 2	44.00	40.29	
	TCP 3	44.00	40.13	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.153	
	TCP 2	N/A	0.152	
	TCP 3	N/A	0.153	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.071	
	TCP 2	N/A	0.070	
	TCP 3	N/A	0.071	

Table 8.7 shows the simulation results for concurrent TCP connections of an ensemble. Here, the overall mean throughput as well as the connection-oriented mean throughput are slightly decreased by approximately 1 % (not significant) and 2 % (significant) if the FS-TCBI controller is used instead of standard TCP. In addition, also the mean fairness between concurrent TCP connections of an ensemble is slightly reduced by the FS-TCBI controller.

Table 8.7: Simulation results of simulation 2, scenario 1 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	31.93	31.54	$=$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	31.12	30.65	$-$
$\overline{I}_f$ of concurrent TCP connections []	0.7943	0.7821	$-$

8.4.2 Unreliable Last Hop

Table 8.8 shows the simulation results for new TCP connections entering an ensemble. The FS-TCBI controller is able to reach a large gain for the overall mean throughput of approximately 40 % compared to standard TCP. The connection-oriented mean throughput of new TCP connections is only slightly negatively affected by the FS-TCBI controller with a throughput degradation of less than 1 % (not significant).

Table 8.8: Simulation results of simulation 2, scenario 2 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	19.67	28.07	$+$
	TCP 2	20.71	28.28	$+$
	TCP 3	19.75	28.26	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	43.10	42.78	$=$
	TCP 2	43.36	42.96	$=$
	TCP 3	43.09	43.03	$=$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	4.17	
	TCP 2	3.00	4.20	
	TCP 3	3.00	4.20	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	31.81	
	TCP 2	44.00	31.73	
	TCP 3	44.00	32.04	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.070	
	TCP 2	N/A	0.070	
	TCP 3	N/A	0.070	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.021	
	TCP 2	N/A	0.021	
	TCP 3	N/A	0.021	

Table 8.9 shows the simulation results for concurrent TCP connections of an ensemble. The FS-TCBI controller reaches a gain of approximately 10 % for the overall mean throughput whereas standard TCP is able to slightly outperform the FS-TCBI controller regarding the connection-oriented mean throughput with a gain of approximately 2 %. Also the mean fairness between concurrent TCP connections of an ensemble is slightly reduced if the FS-TCBI controller is used.

Table 8.9: Simulation results of simulation 2, scenario 2 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	27.32	30.06	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	39.65	38.81	$-$
$\overline{I}_f$ of concurrent TCP connections []	0.8342	0.8263	$-$

8.5 Simulation 3: TCP NewReno, RTT $\geq$ 100 ms

8.5.1 Reliable Last Hop

Table 8.10 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, for new TCP connections the FS-TCBI controller is able to increase the overall mean throughput by approximately 12 % and the connection-oriented mean throughput by approximately 10 %.

Table 8.10: Simulation results of simulation 3, scenario 1 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	30.45	33.94	+
	TCP 2	30.52	33.93	+
	TCP 3	30.13	34.02	+
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	31.11	34.17	+
	TCP 2	31.20	34.26	+
	TCP 3	31.16	34.43	+
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	8.95	
	TCP 2	3.00	9.08	
	TCP 3	3.00	9.01	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.25	
	TCP 2	44.00	40.25	
	TCP 3	44.00	40.32	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.190	
	TCP 2	N/A	0.194	
	TCP 3	N/A	0.193	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.077	
	TCP 2	N/A	0.078	
	TCP 3	N/A	0.078	

Table 8.11 shows the simulation results for concurrent TCP connections of an ensemble. The FS-TCBI controller is able to slightly increase the overall mean throughput by approximately 1 % (not significant) and the connection-oriented mean throughput by approximately 2 % (significant) compared to standard TCP. But the mean fairness between concurrent TCP connections of an ensemble is slightly reduced if the FS-TCBI controller is used.

Table 8.11: Simulation results of simulation 3, scenario 1 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	27.51	27.83	$=$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	25.97	26.59	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7907	0.7828	$-$

8.5.2 Unreliable Last Hop

Table 8.12 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, the FS-TCBI controller is able to largely increase the overall mean throughput of new TCP connections by approximately 35 %. In addition, also the connection-oriented mean throughput of these TCP connections is slightly increased by 3 % by the FS-TCBI controller.

Table 8.12: Simulation results of simulation 3, scenario 2 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	16.94	22.64	$+$
	TCP 2	16.89	22.92	$+$
	TCP 3	16.96	22.98	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	30.38	31.31	$+$
	TCP 2	30.18	31.34	$+$
	TCP 3	30.41	31.25	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	4.37	
	TCP 2	3.00	4.32	
	TCP 3	3.00	4.27	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	31.04	
	TCP 2	44.00	30.88	
	TCP 3	44.00	30.83	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.106	
	TCP 2	N/A	0.107	
	TCP 3	N/A	0.107	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.025	
	TCP 2	N/A	0.025	
	TCP 3	N/A	0.025	

Table 8.13 shows the simulation results for concurrent TCP connections of an ensemble. For the overall mean throughput the FS-TCBI controller reaches a gain of approximately 8 % compared to standard TCP. Also the connection-oriented mean throughput is slightly increased by approximately 1 % if the FS-TCBI controller is used. But the mean fairness between concurrent TCP connections of an ensemble is slightly reduced by the FS-TCBI controller.

Table 8.13: Simulation results of simulation 3, scenario 2 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	21.48	23.29	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	27.48	27.76	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.8402	0.8328	$-$

8.6 Simulation 4: TCP NewReno, RTT $\geq$ 500 ms

8.6.1 Reliable Last Hop

Table 8.14 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, for new TCP connections the FS-TCBI controller is able to increase both the overall mean throughput and the connection-oriented mean throughput by approximately 14 %.

Table 8.14: Simulation results of simulation 4, scenario 1 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	13.16	14.84	$+$
	TCP 2	12.97	15.05	$+$
	TCP 3	13.10	14.92	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	12.28	13.95	$+$
	TCP 2	12.14	14.02	$+$
	TCP 3	12.22	13.96	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	9.50	
	TCP 2	3.00	9.55	
	TCP 3	3.00	9.54	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	42.05	
	TCP 2	44.00	42.00	
	TCP 3	44.00	42.03	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.557	
	TCP 2	N/A	0.558	
	TCP 3	N/A	0.557	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.126	
	TCP 2	N/A	0.126	
	TCP 3	N/A	0.126	

Table 8.15 shows the simulation results for concurrent TCP connections of an ensemble. Both the overall mean throughput and the connection-oriented mean throughput are slightly increased by approximately 2 % and 3 %, respectively, by the FS-TCBI controller compared to standard TCP. But the mean fairness between concurrent TCP connections of an ensemble is slightly reduced by the FS-TCBI controller.

Table 8.15: Simulation results of simulation 4, scenario 1 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	11.65	11.88	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	10.45	10.82	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7757	0.7672	$-$

8.6.2 Unreliable Last Hop

Table 8.16 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, the FS-TCBI controller reaches an approximately 19 % higher overall mean throughput and an approximately 6 % higher connection-oriented mean throughput.

Table 8.16: Simulation results of simulation 4, scenario 2 (new TCP connections)

		no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	6.30	7.50	$+$
	TCP 2	6.31	7.48	$+$
	TCP 3	6.29	7.46	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	7.78	8.27	$+$
	TCP 2	7.80	8.29	$+$
	TCP 3	7.79	8.22	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	4.69	
	TCP 2	3.00	4.70	
	TCP 3	3.00	4.68	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	28.90	
	TCP 2	44.00	28.94	
	TCP 3	44.00	28.82	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.498	
	TCP 2	N/A	0.497	
	TCP 3	N/A	0.498	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.066	
	TCP 2	N/A	0.066	
	TCP 3	N/A	0.066	

Table 8.17 shows the simulation results for concurrent TCP connections of an ensemble. For these connections, the FS-TCBI controller is able to outperform standard TCP with an approximately 11 % higher overall mean throughput and an approximately 7 % higher connection-oriented mean throughput. But the mean fairness between concurrent TCP connections of an ensemble is slightly reduced if the FS-TCBI controller is used.

Table 8.17: Simulation results of simulation 4, scenario 2 (concurrent TCP connections)

	no FS-TCBI	FS-TCBI	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	6.25	6.91	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	6.38	6.85	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.8571	0.8485	$-$

8.7 Summary and Discussion

The following Figures 8.1 to 8.5 show the mean values of the considered performance metrics dependent on the minimum round trip time. In addition, for every mean value its 99 % confidence interval is stated.

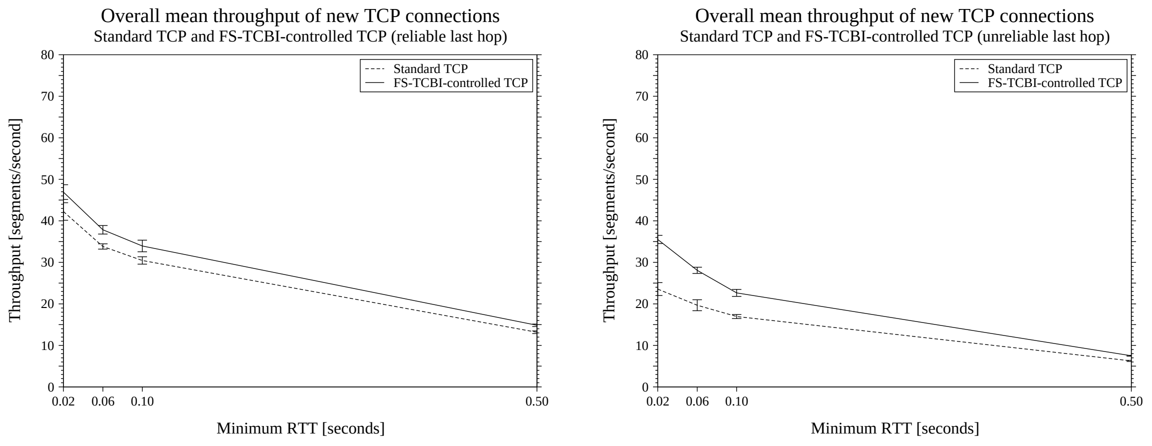


Figure 8.1: Overall mean throughput of new standard or FS-TCBI-controlled TCP connections

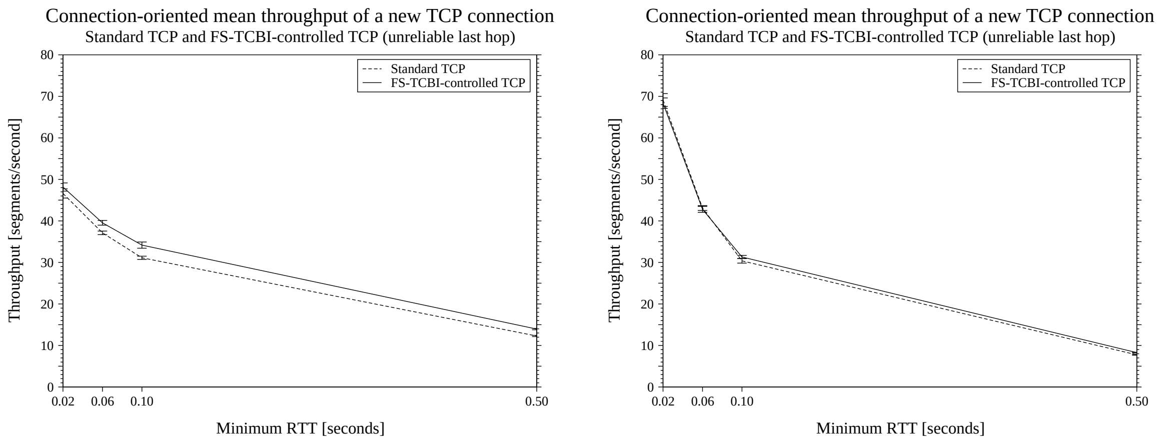


Figure 8.2: Connection-oriented mean throughput of a new standard or FS-TCBI-controlled TCP connection

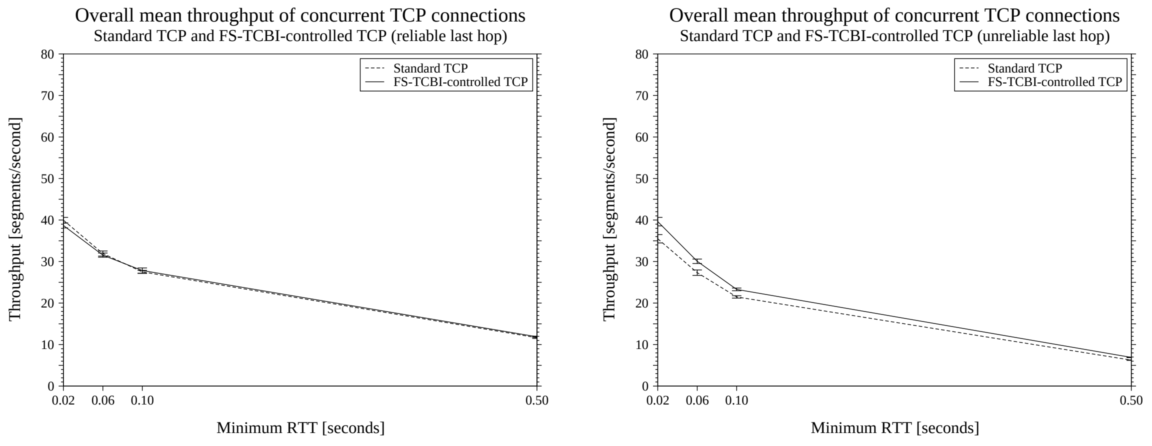


Figure 8.3: Overall mean throughput of concurrent standard or FS-TCBI-controlled TCP connections

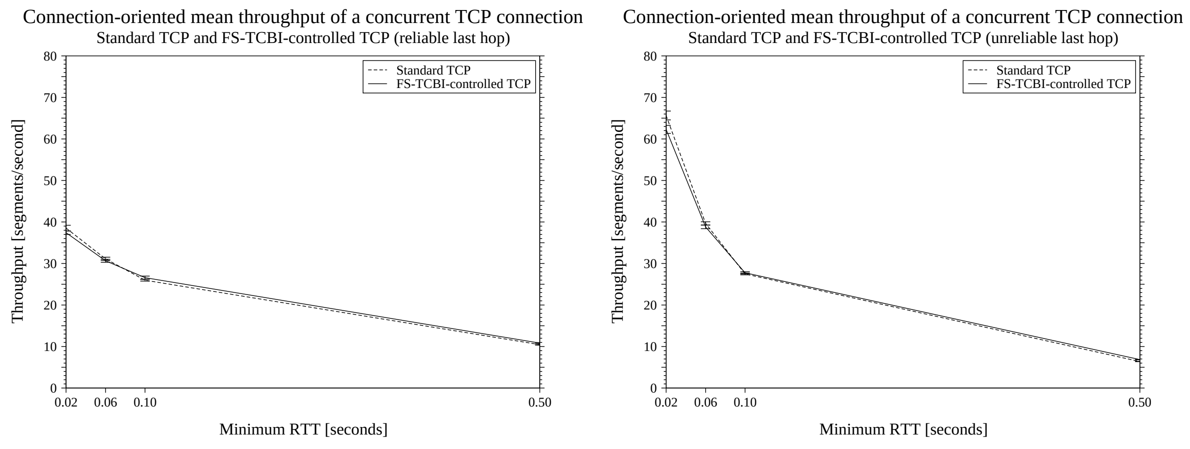


Figure 8.4: Connection-oriented mean throughput of a concurrent standard or FS-TCBI-controlled TCP connection

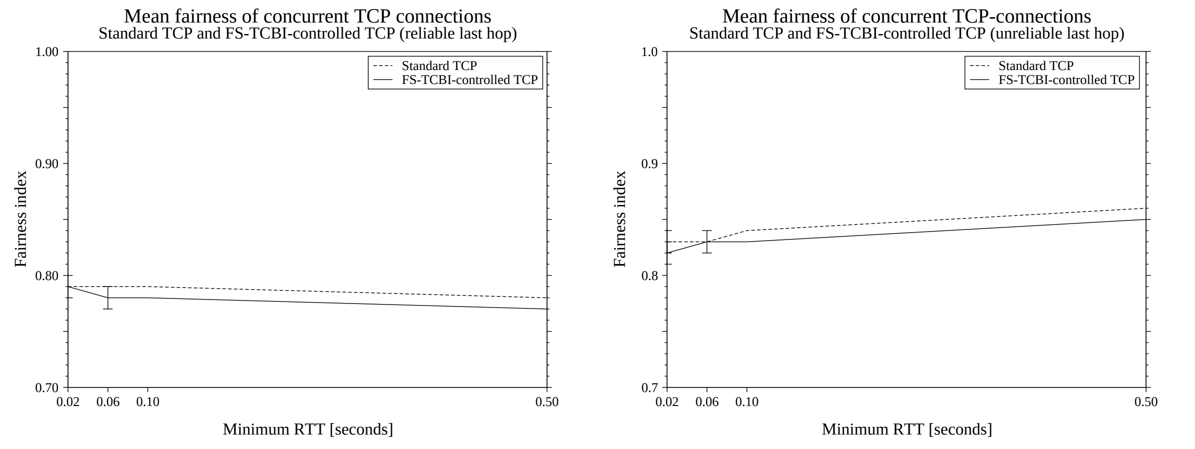


Figure 8.5: Mean fairness index of concurrent standard or FS-TCBI-controlled TCP connections

It can be seen that the benefits and disadvantages of a one-time network-information sharing approach depends to a large degree on the simulation scenario. To summarize:

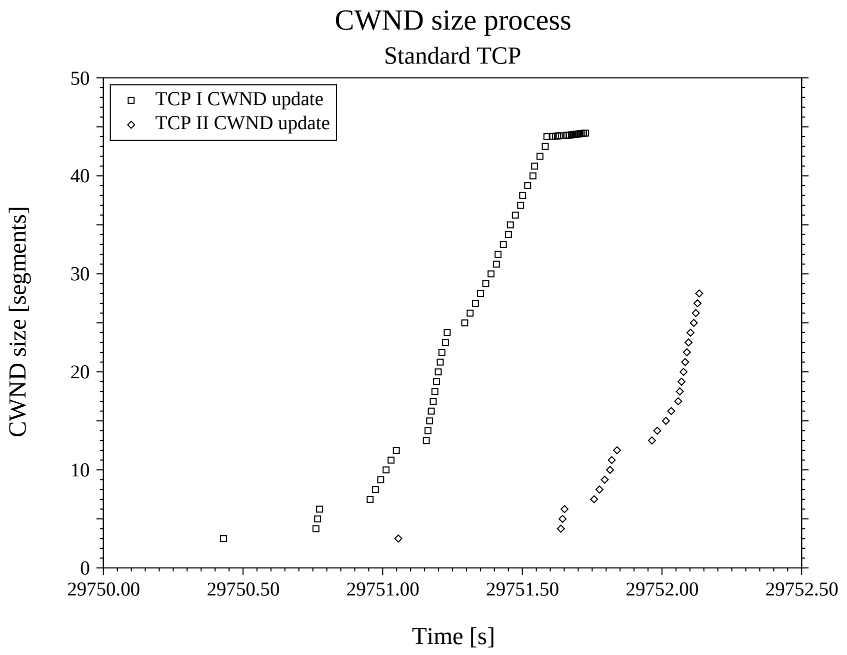
- Reliable last hop scenario: In all considered simulation scenarios both the overall mean throughput and the connection-oriented mean throughput of new FS-TCBI-controlled TCP connections are increased compared to standard TCP. The performance gain of the FS-TCBI controller for both throughput metrics is positively correlated with the minimum round trip time: the gain for the overall mean throughput slightly increases from 10 % to 14 %, the gain for the connection-oriented mean throughput is in the range from 3 % up to 14 %. Thus, a new TCP connection can expect a mean increase in its throughput if the sending end system is equipped with FS-TCBI. For concurrent TCP connections, a different result can be observed. Here, the FS-TCBI-controlled TCP connections reaches slightly larger overall and connection-oriented mean throughputs only in cases with minimum round trip times larger than 60 ms. For lower minimum round trip times both throughput metrics are slightly decreased if the FS-TCBI controller is used.

  What is very interesting is that the mean fairness between FS-TCBI-controlled TCP connections is always slightly lower than the mean fairness between standard TCP connections. But this rather unexpected result can be explained by considering the method to calculate the throughputs of concurrent TCP connections in more detail: The throughput metrics for concurrent TCP connections consider only TCP connections with an observed throughput greater than zero, i.e., only those TCP connections are considered which have sent some TCP segments and have received the associated TCP acknowledgments during the period of time where the TCP connections are active in parallel. Since an existing standard TCP connection is not automatically reduced in its congestion window by a starting new TCP connection, it can send its remaining TCP segments, receive the associated TCP acknowledgments, finishes the connection with a high throughput, and closes the observation time for concurrent TCP connections. In this period of time, it can happen that the starting new TCP connection does not receive a single TCP acknowledgment, i.e., it achieves a throughput of zero and is therefore not considered in the calculations of the throughput metrics of concurrent TCP connections. In the same case, concurrent FS-TCBI-controlled TCP connections receive always throughputs greater than zero but mostly lower than the throughput of the existing standard TCP connection. Therefore, the chosen throughput metrics for concurrent TCP connections tend to prefer standard TCP connections, i.e., they are sometimes a little bit unfair to FS-TCBI-controlled TCP connections.

In the considered simulation scenarios with a reliable last hop, the mean initial congestion window is nearly independent of the minimum round trip time. In the mean, TCP connections with a maximum size of 9 segments will largely benefit from the FS-TCBI controller, since these TCP connections can transfer all their segments in a single (paced) burst.

Figures 8.6 and 8.7 show some typical processes of the congestion window and the sequence number for either two standard TCP NewReno connections (with a fairness index of $I_f = 0.6098$) or two FS-TCBI-controlled TCP NewReno connections ($I_f = 0.7461$) of an ensemble traversing a network path with a minimum round trip time of 100 ms and a reliable last hop. In Figure 8.6, the concurrent TCP connections are separately controlled and start with different congestion windows into their time of concurrency. Figure 8.7 shows FS-TCBI-controlled concurrent TCP connections which start with the same congestion windows into their time of concurrency. For the new TCP connection entering the ensemble it can be seen how the pacing mechanism of the FS-TCBI controller is able to avoid a bursty sending behavior at the cost of a reduced throughput during the starting phase of the new TCP connection.

- Unreliable last hop scenario: In all considered simulation scenarios the overall mean throughput of new FS-TCBI-controlled TCP connections is (largely) increased compared to standard TCP. The performance gain of FS-TCBI is negatively correlated with the minimum round trip time and reaches values from 48 % down to 19 %. But the connection-oriented mean throughput of new TCP connections is only slightly increased by the FS-TCBI controller if the minimum round trip time is larger than 60 ms. For lower minimum round trip times standard TCP reaches slightly larger values. The connection-oriented overall mean throughput of concurrent TCP connections is always increased by the FS-TCBI controller compared to standard TCP. Here, the performance gain of FS-TCBI is 12 %, 10 %, 8 %, and 11 % for the different minimum round trip times.



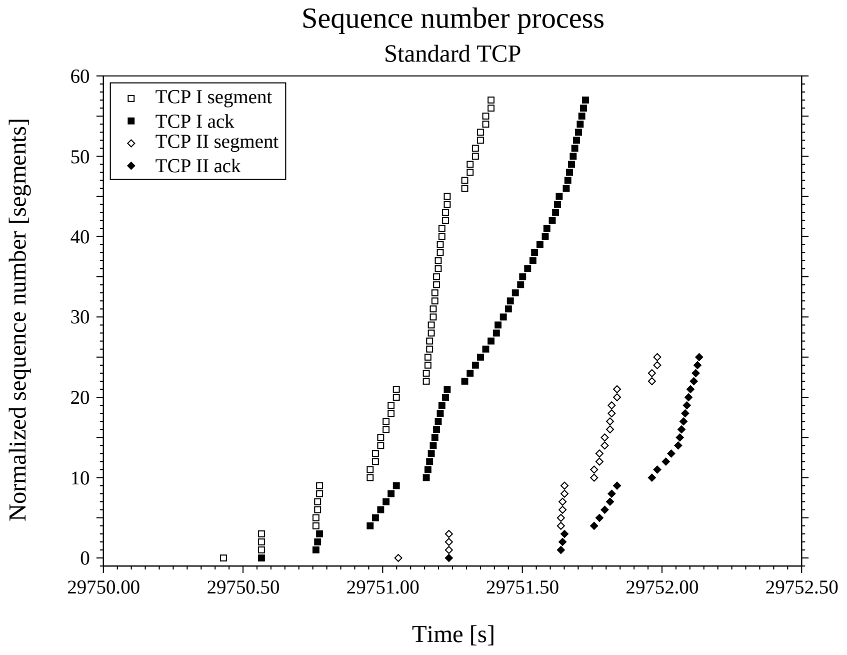
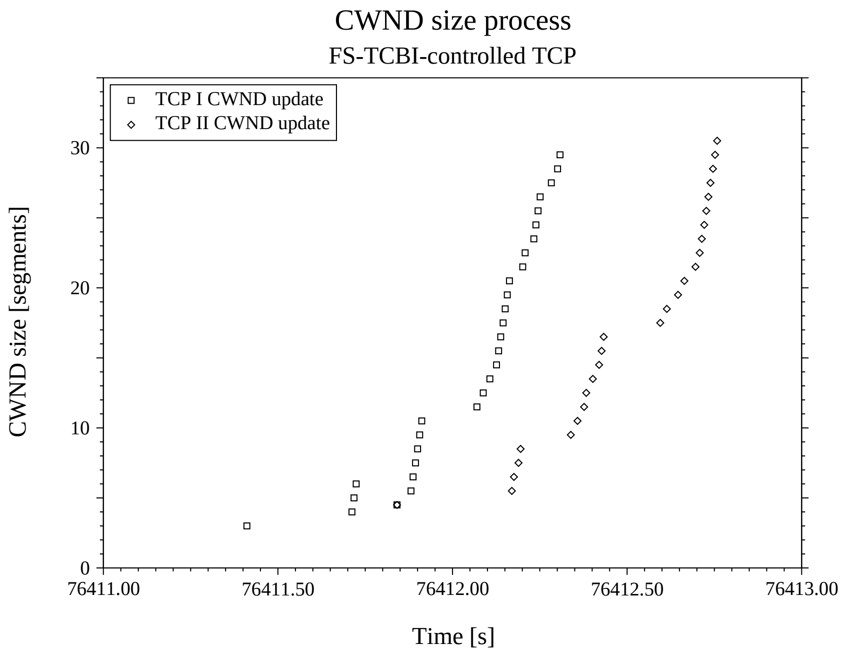


Figure 8.6: Standard TCP connections over a reliable last hop — □ and ◇ represent the first and second TCP connection of a virtual ensemble

But the connection-oriented mean throughput of these TCP connections is only slightly increased by the FS-TCBI controller if the minimum round trip time is above 60 ms. For lower minimum round trip times standard TCP reaches slightly larger values. Similar to the results obtained in the scenarios with a reliable last hop, the fairness between FS-TCBI controlled TCP connections is slightly decreased independent of the considered minimum round trip time. The main reason for this result is based on the above explained property of the throughput calculation that sometimes prefer standard TCP connections.



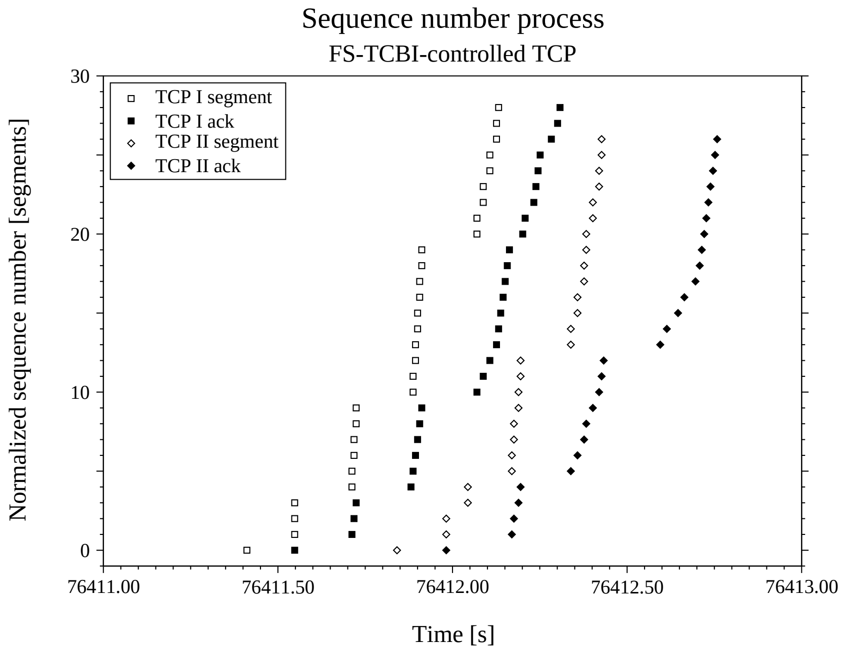
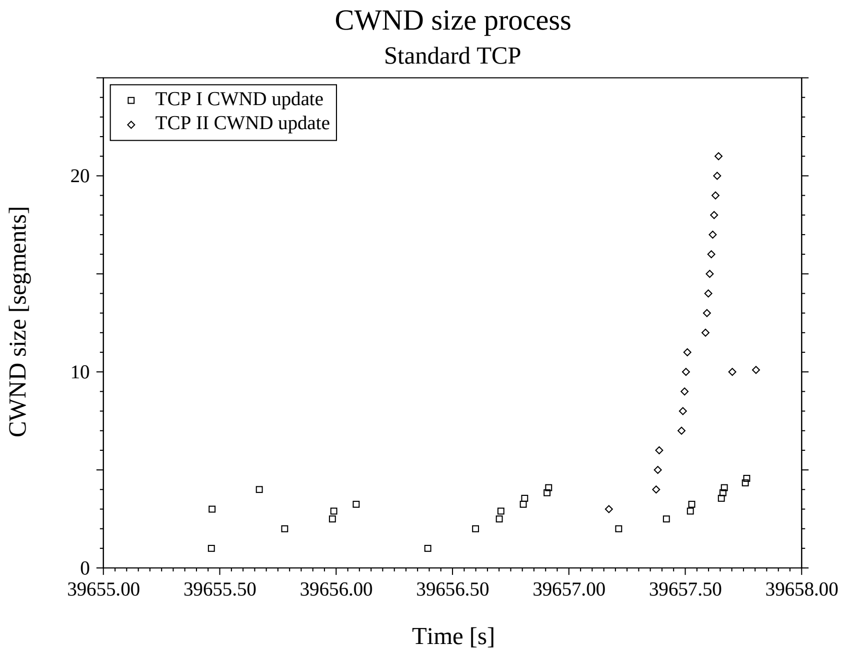


Figure 8.7: FS-TCBI-controlled TCP connections over a reliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble

In the considered simulation scenarios with an unreliable last hop, the mean initial congestion window is nearly independent of the minimum round trip time. In the mean, TCP connections with a maximum size of 4 segments will largely benefit from the FS-TCBI controller, since these TCP connections can transfer all their segments in a single (paced) burst.

Figures 8.8 and 8.9 show some typical processes of the congestion window and the sequence number for either two standard TCP NewReno connections ($I_f = 0.8118$) or two FS-TCBI-controlled TCP NewReno connections ($I_f = 0.6269$) of an ensemble traversing a network path with a mini-



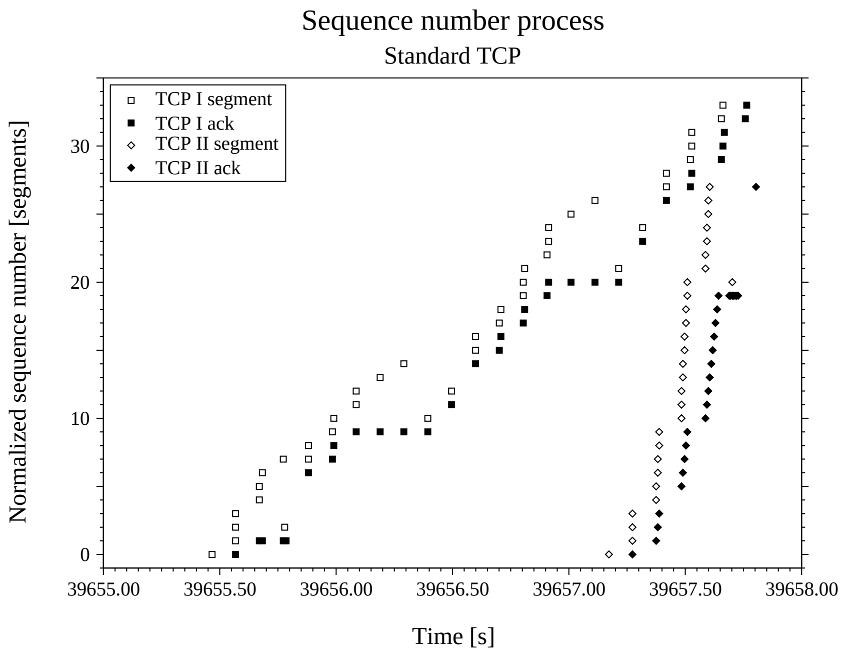
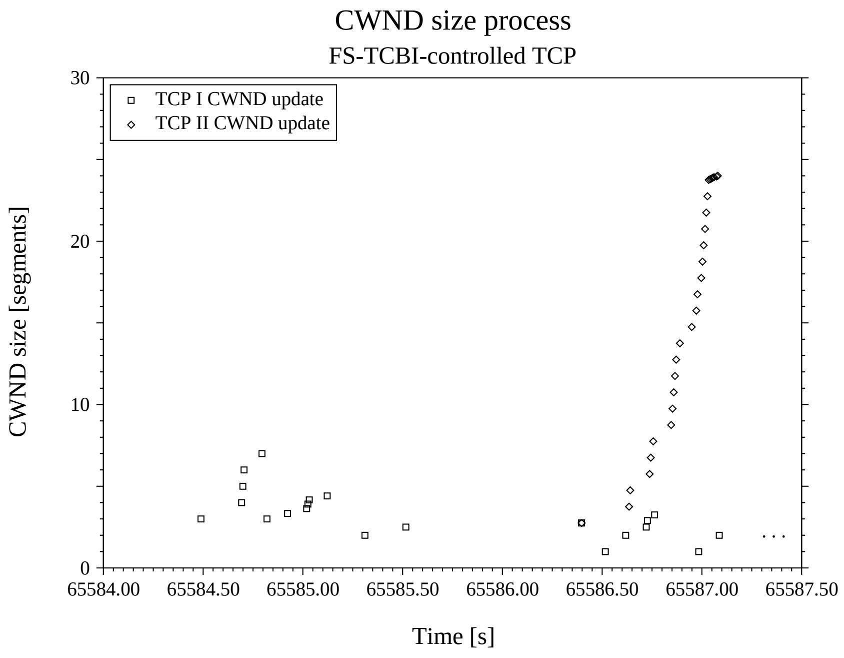


Figure 8.8: Standard TCP connections over an unreliable last hop — □ and ◇ represent the first and second TCP connection of a virtual ensemble

mum round trip time of 100 ms and an unreliable last hop. In Figure 8.8, the concurrent TCP connections are separately controlled and start with different congestion windows during their time of concurrency. Figure 8.9 shows FS-TCBI-controlled concurrent TCP connections. Although these two TCP connections start with the same congestion windows into their time of concurrency they obtain very different congestion windows during this time period due to different error patterns they experience. Thus, a one-time network-information sharing approach like the FS-TCBI controller is not able to handle packet losses much better than standard TCP does regarding the



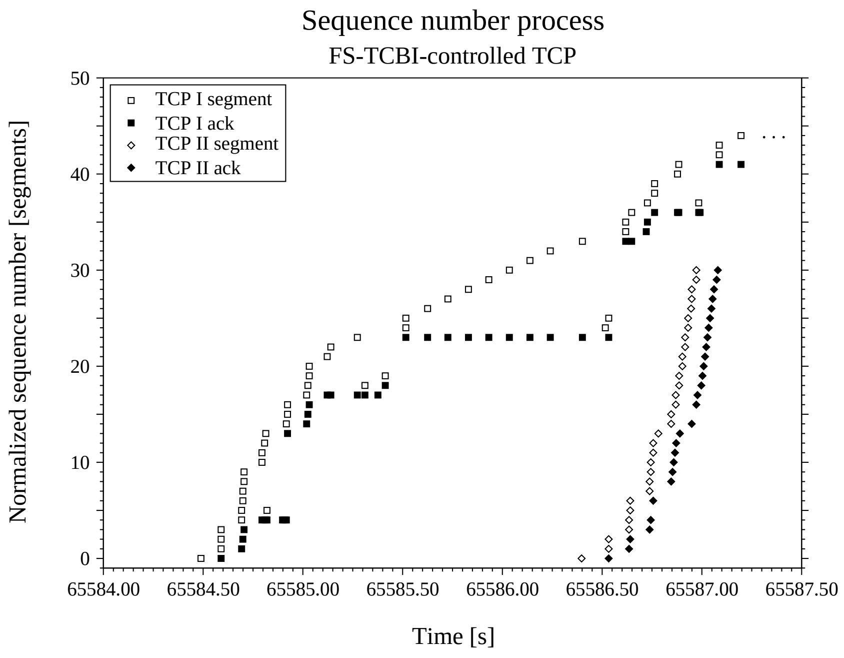
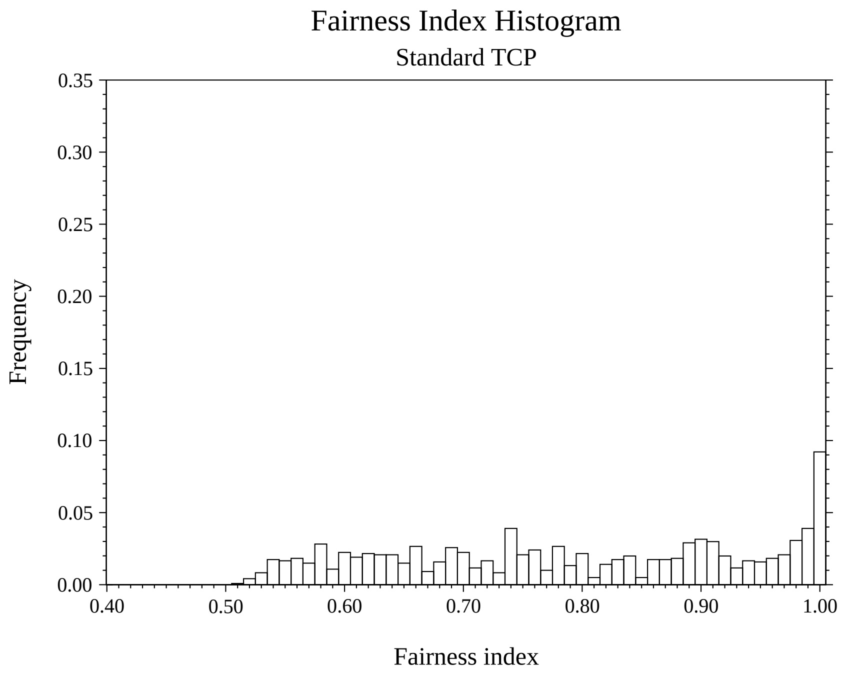


Figure 8.9: FS-TCBI-controlled TCP connections over an unreliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble

throughput of and fairness in an ensemble.

It is remarkable that in the first two simulations with an unreliable last hop the connection-oriented mean throughput (not the overall mean throughput!) of some considered TCP connections is higher than in the simulation scenarios with a reliable last hop. But this—at first astonishing—result can be easily explained: The background-traffic TCP connections with receivers located in a receiving end system in the receiver LAN are often affected by packet losses in the unreliable last hop. Due to the TCP congestion-control algorithms these background-traffic TCP connections reach a smaller overall



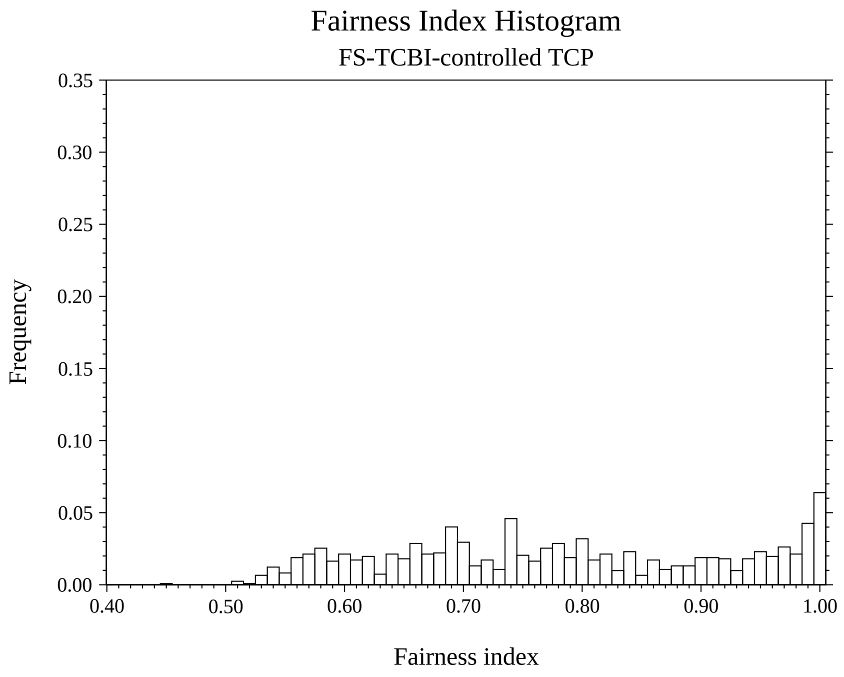
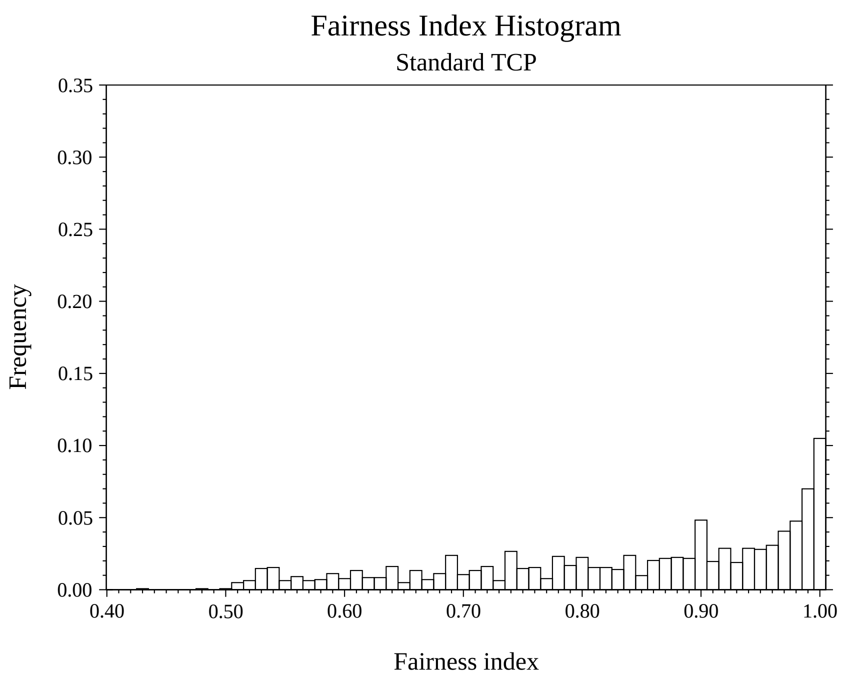


Figure 8.10: Fairness index histogram for concurrent standard or FS-TCBI-controlled TCP connections over a reliable last hop

congestion window, allocate less bandwidth, and produce a substantially lower load in the receiver LAN. The other TCP connections with receivers in the receiver LAN are also affected by packet losses in the unreliable last hop. But from a single TCP connection's point of view, the lower load in the shared medium of the receiver LAN can sometimes compensate for and even overcompensate for the in general negative influence of packet losses in an unreliable last hop on the mean throughput of a single TCP connection. For example, some of the TCP connections are not affected at all by packet losses during



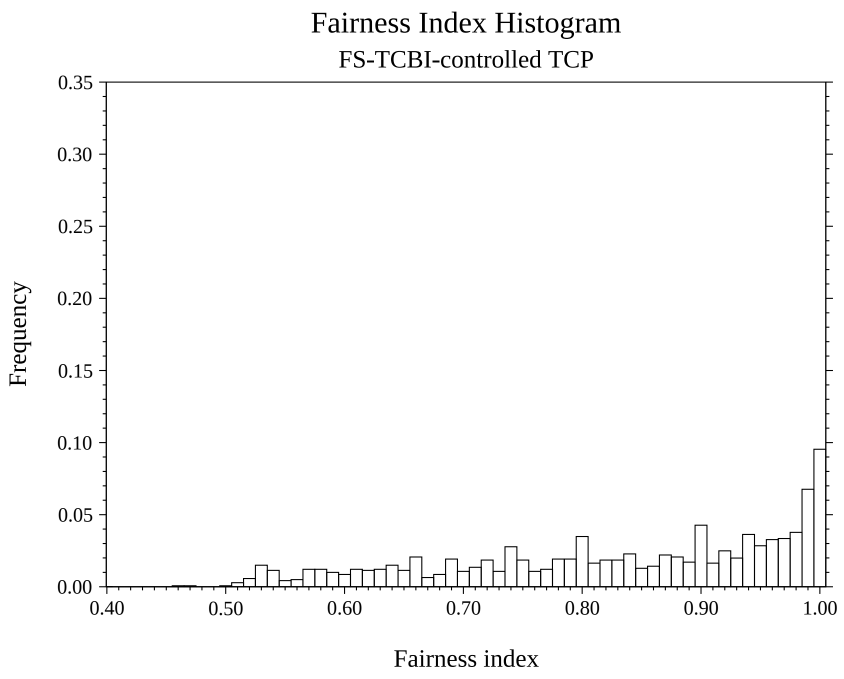
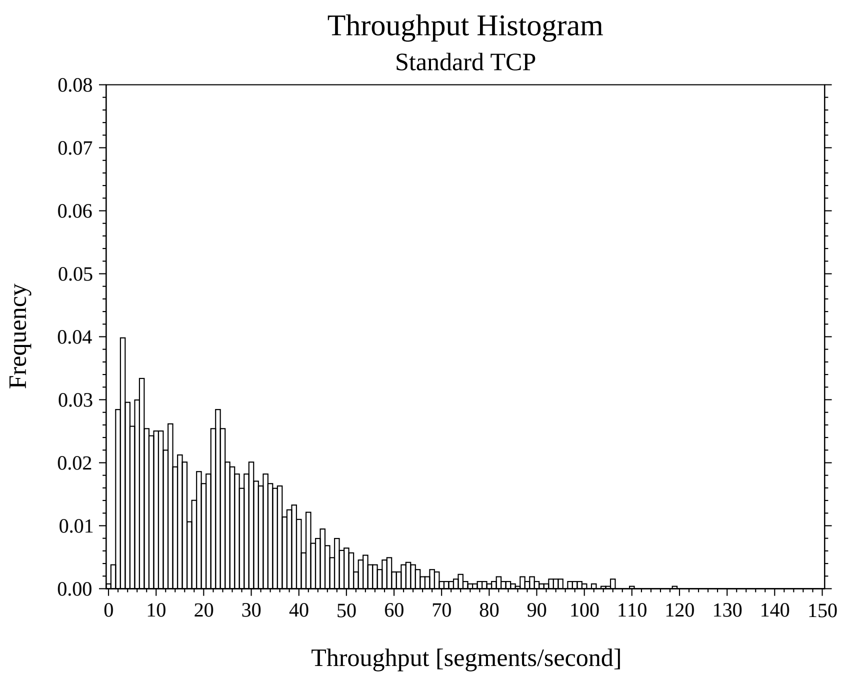


Figure 8.11: Fairness index histogram for concurrent standard or FS-TCBI-controlled TCP connections over an unreliable last hop

their whole lifetime. These TCP connections can highly benefit from the lower load in the receiver LAN and reach a higher connection-oriented mean throughput. Therefore, the observed higher connection-oriented mean throughput in the unreliable last hop scenario is only based on the lower load in the receiver LAN.

Figures 8.10 and 8.11 show the histograms of the fairness index that concurrent standard or FS-TCBI-controlled TCP connections reach in the simulations with a minimum round trip time of 100 ms



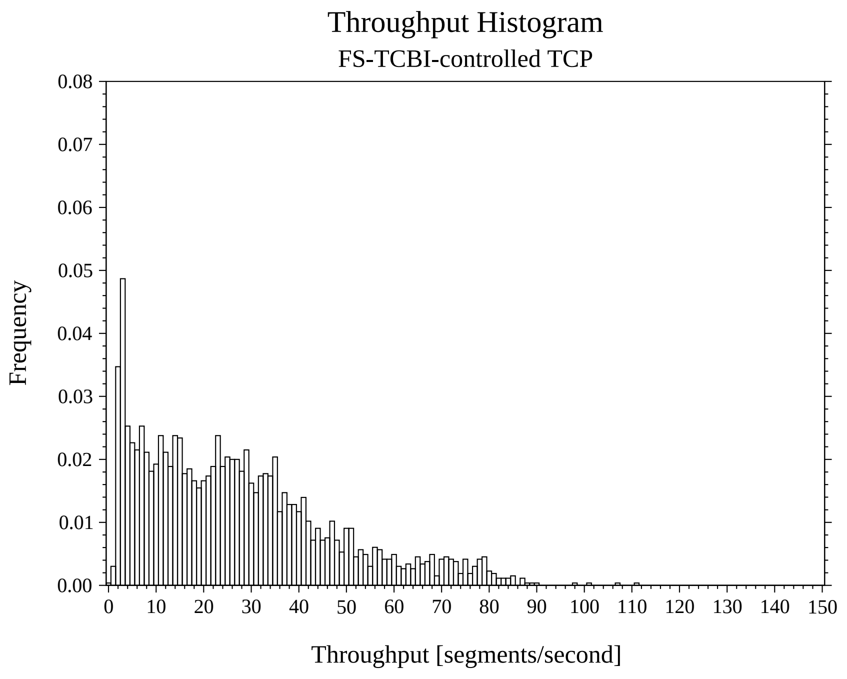
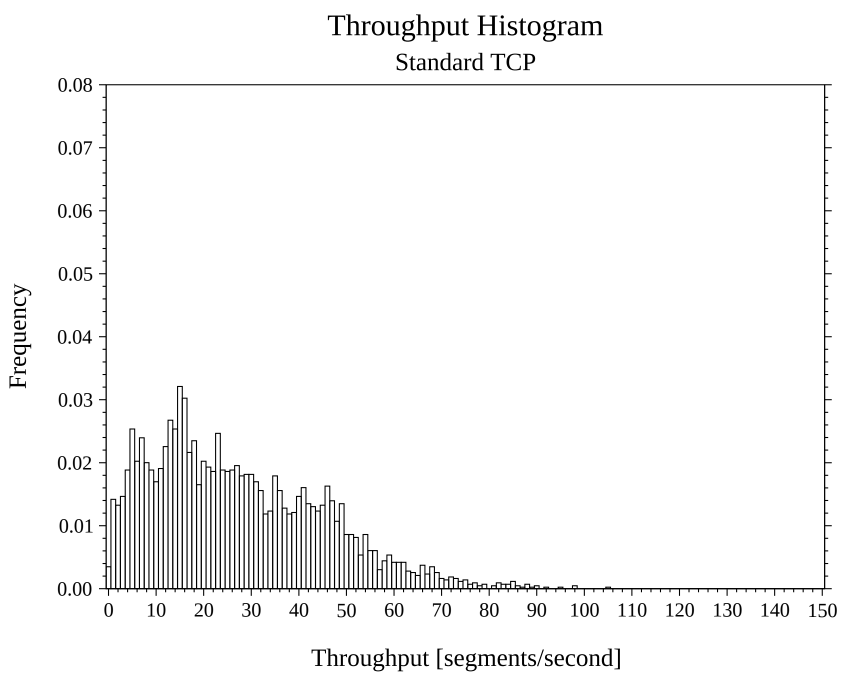


Figure 8.12: Throughput histogram for concurrent standard or FS-TCBI-controlled TCP connections over a reliable last hop

regarding the two last hop scenarios. Only minor differences in these fairness-index histograms can be observed between the different approaches.

Figures 8.12 and 8.13 show the histograms of the throughput that concurrent standard or FS-TCBI-controlled TCP connections reach in the simulations with a minimum round trip time of 100 ms regarding the two last hop scenarios. Compared to FS-TCBI-controlled TCP connections, it is more likely that standard TCP connections reach lower as well as higher throughputs. In contrast, the FS-TCBI controller



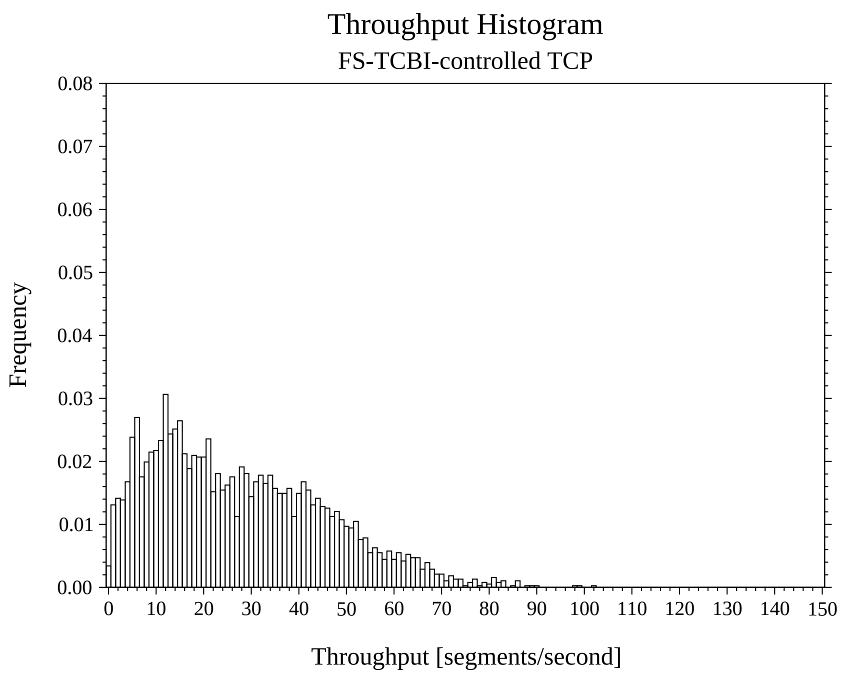


Figure 8.13: Throughput histogram for concurrent standard or FS-TCBI-controlled TCP connections over an unreliable last hop

is able to slightly increase the probability that TCP connections obtain mean throughputs.

In Table 8.18, the performance of the FS-TCBI controller derived from the simulation results is summarized. A '+++', '++', or '+' denotes that the FS-TCBI controller achieves a huge gain ($\geq$ 50 %), a large gain ($\geq$ 20 %), or a gain ($<$ 20 %) compared to the standard TCP controller and with respect to the performance metric shown in this row; a '=' denotes that no significant difference between the standard TCP and the FS-TCBI controller can be observed; and a '-' denotes that standard TCP achieves a slight gain compared to the FS-TCBI controller.

Table 8.18: Performance of the FS-TCBI controller compared to standard TCP — $\overline{TPO}_{\text{new}}$ is the overall mean throughput of new TCP connections, $\overline{TPC}_{\text{new}}$ is the connection-oriented mean throughput of new TCP connections, $\overline{TPO}_{\text{concurrent}}$ is the overall mean throughput of concurrent TCP connections, $\overline{TPC}_{\text{concurrent}}$ is the connection-oriented mean throughput of concurrent TCP connections, and $\overline{I}_{f,\text{concurrent}}$ is the mean fairness index for concurrent TCP connections ($^* =$ results differ for each of the considered minimum round trip times 20 ms, 60 ms, 100 ms, and 500 ms)

	Reliable Last Hop	Unreliable last hop
$\overline{TPO}_{\text{new}}$	+	++,++,++,+*
$\overline{TPC}_{\text{new}}$	+	-,=,+,+*
$\overline{TPO}_{\text{concurrent}}$	-,=,=,+*	+
$\overline{TPC}_{\text{concurrent}}$	-,-,+,+*	-,-,+,+*
$\overline{I}_{f,\text{concurrent}}$	=,-,-,-*	=,-,-,-*

8.8 Conclusion and Outlook

A one-time network-information sharing approach like FS-TCBI is able to increase the throughput of new and concurrent TCP connections in an ensemble under various network conditions. Solely for low minimum round trip times in the network path the throughput of concurrent TCP connections is slightly reduced compared to standard TCP. In addition, FS-TCBI is not able to increase the fairness between concurrent TCP connections, in fact, the fairness between FS-TCBI-controlled TCP connections is slightly reduced compared to standard TCP. It can be further observed that the throughput gain for new TCP connections entering a FS-TCBI-controlled ensemble is based on a fairness reduction between all TCP connections in the ensemble combined with a lower ensemble throughput compared to more sophisticated and more complex approaches like EFCM. Thus, one-time network-information sharing approaches like FS-TCBI are not well-balanced regarding their impact on the different performance metrics. In anticipation of the simulation results of the common congestion controller EFCM shown in the next chapter, the ensemble throughput of FS-TCBI is (much) lower than the ensemble throughput reached with EFCM. This is the outcome of the FS-TCBI control algorithms that are not able to keep the best-case behavior of standard TCP connections during their whole lifetimes (cf. Chapter 6). Hence, the performance gain of FS-TCBI and other one-time network-information sharing approaches is inherently limited.

Chapter 9

Performance Evaluation of EFCM by Simulations

9.1 Introduction

Simulations using the simulation model described in Section 7.3 are performed with the standard TCP controller and with the EFCM controller in the (EFCM-capable) sending end system. The last hop in this simulation model is either reliable or has a fixed packet loss rate (PLR) of 5 % in the receiver LAN. In addition, the minimum round trip time is varied to investigate the performance of the different controllers in the sending end system dependent on this network parameter. This is done by changing the propagation delay between the two routers in the simulated network topology.

In this chapter, the simulation results for the two controllers considering several simulation scenarios with varied minimum round trip times and two last hop scenarios are shown in detail. Section 9.2 contains a general overview about the performed simulations. The following Sections 9.3 to 9.6 show the simulation results for the different simulation scenarios in detail. A summary and discussion of these simulation results can be found in Section 9.7. Finally, conclusions and an outlook to future research in the context of common congestion-control approaches are given in Section 9.8.

9.2 Overview

For both last hop scenarios the different TCP controllers (standard TCP/no EFCM, EFCM) are investigated for a simulated time of 250000 s in each simulation run to reach more expressive values for the mean performance metrics. This means that, for example, in the case with a minimum round trip time of 100 ms approximately 21000 TCP connections in the reliable last hop scenario and approximately 20500 TCP connections in the unreliable last hop scenario starting at the (EFCM-capable) sending end system can be observed at an average during the simulated time. Only some of them, i.e., those TCP connections which have concurrent TCP connections, are controlled or could be controlled by the new one-time network-information sharing and common congestion-control mechanisms provided by the EFCM controller. In the simulation model and with the chosen traffic load model the percentage of having concurrent TCP connections is relatively low. An average computation over all simulations with a minimum round trip time of 100 ms shows that approximately 8.5 % of the new TCP connections in the reliable last hop scenarios and approximately 13.9 % of the new TCP connections in the unreliable last hop scenarios are controlled or could be controlled by the EFCM. This amount of EFCM-controllable TCP connections increases with the minimum round trip time between the TCP senders and the TCP receivers to values up to 18.0 % for the reliable last hop and 31.1 % for the unreliable last hop in the case with a minimum round trip time of 500 ms.

In all simulations with the EFCM controller and with the observed occurrence of having concurrent TCP connections the mean throughput of the background TCP connections is not negatively affected by these EFCM-controlled TCP connections. Starting from these simulation results, it can be assumed that EFCM-controlled TCP connections are no more aggressive to the network than standard TCP connections. This important result can be also concluded from the mean packet loss rates in the bottleneck router observed during the whole simulated time of a single simulation run and averaged over all simulation runs of a simulation scenario. The following Table 9.1 compares the mean PLRs observed in the simulation scenarios with standard TCP connections with the mean PLRs observed in the simulation scenarios with EFCM-controlled TCP connections.

Table 9.1: Packet loss rates of standard TCP and EFCM-controlled TCP connections

Scenario	no EFCM	EFCM	Difference (absolute, in %)
RTT $\geq$ 20 ms, reliable last hop	0.003263	0.003256	$+0.000007.\quad +\ 0.2150$
RTT $\geq$ 60 ms, reliable last hop	0.003257	0.003257	$+0.000000,\quad +\ 0.0000$
RTT $\geq$ 100 ms, reliable last hop	0.003275	0.003265	$+0.000010,\quad +\ 0.3063$
RTT $\geq$ 500 ms, reliable last hop	0.003225	0.003228	$-0.000003,\quad -\ 0.0929$
RTT $\geq$ 20 ms, unreliable last hop	0.000028	0.000027	$+0.000001,\quad +\ 3,7037$
RTT $\geq$ 60 ms, unreliable last hop	0.000028	0.000025	$+0.000003,\quad +12.0000$
RTT $\geq$ 100 ms, unreliable last hop	0.000027	0.000028	$-0.000001,\quad -\ 3.5714$
RTT $\geq$ 500 ms, unreliable last hop	0.000027	0.000028	$-0.000001,\quad -\ 3.5714$

In Chapter 6 (cf. [78]), it has been analytically shown that the aggressiveness of the algorithms of the EFCM controller regarding the congestion window and the slow start threshold is comparable to the aggressiveness of standard TCP in its best case where the above mentioned control variables are fairly shared between concurrent TCP connections in a virtual ensemble. Since this best case is rarely reached by concurrent standard TCP connections in the simulations, the simulation results regarding the mean PLRs show that the aggressiveness of EFCM-controlled TCP connections is also comparable to the aggressiveness of standard TCP connections in the mean case, at least for the typical packet loss rates in the bottleneck router considered in the simulations.

In the following Tables 9.2 to 9.16, TCP 1, TCP 2, and TCP 3 are EFCM-controllable TCP connections of the (EFCM-capable) sending end system transferring WWW data. The simulation results of each simulation scenario shown in these tables are averages over twelve independent simulation runs, each of these runs is performed for a large simulated time of 250000 seconds. The transient analysis of the simulations (cf. Appendix A) shows that the throughputs of the TCP connections in the (EFCM-capable) sending end system have no transient phase. Since all considered performance metrics depend on the observed throughputs of these TCP connections, all throughput observations during the whole simulated time can be used to calculate the mean values of the different performance metrics.

Both stated mean throughput metrics ($\overline{TPO}$, $\overline{TPC}$) of the EFCM-controllable TCP connections are expressed in segments per second. For the new TCP connections entering an ensemble also the mean initial congestion window ($\overline{\text{ICWND}}$), the mean initial slow start threshold ($\overline{\text{ISSTHRESH}}$), the mean initial smoothed round trip time ($\overline{\text{ISRTT}}$), and the mean initial round trip time variance ($\overline{\text{IRTTVAR}}$) of their senders are shown. For the concurrent TCP connections of an ensemble both mean throughput metrics ($\overline{TPO}$, $\overline{TPC}$) and the mean fairness index ($\overline{I}_f$) are shown.

The last column Δ in these tables denotes whether the simulation results are statistically significantly different or not for a given confidence level. A '+' or '-' denotes that the EFCM controller or standard TCP is significantly better. A '=' means that with the simulation results no significant difference between the EFCM and the standard TCP controller can be concluded. The details of the statistical evaluation of the simulation results can be found in Appendix B of this dissertation.

For senders of standard TCP connections the mean initial smoothed round trip time and the mean initial round trip time variance are not applicable (N/A) for the computation of the initial retransmission timeout timer, i.e., the initial retransmission timeout timer is set to the fixed standard value.

9.3 Simulation 1: TCP NewReno, RTT $\geq$ 20 ms

9.3.1 Reliable Last Hop

Table 9.2 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, for new TCP connections the overall mean throughput is slightly decreased by 4 % by the EFCM controller. But the connection-oriented mean throughput is increased by approximately 8 % if the EFCM controller is used.

Table 9.2: Simulation results of simulation 1, scenario 1 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	42.25	41.48	$=$
	TCP 2	42.30	39.88	$-$
	TCP 3	42.12	40.37	$-$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	46.61	50.68	$+$
	TCP 2	46.74	49.95	$+$
	TCP 3	46.89	50.22	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	34.52	
	TCP 2	3.00	32.94	
	TCP 3	3.00	35.01	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	70.35	
	TCP 2	44.00	68.56	
	TCP 3	44.00	71.24	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.121	
	TCP 2	N/A	0.119	
	TCP 3	N/A	0.122	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.079	
	TCP 2	N/A	0.077	
	TCP 3	N/A	0.080	

Table 9.3 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller reaches a slight performance degradation of approximately 3 % for the overall mean throughput and a performance gain of approximately 8 % for the connection-oriented mean throughput. The fairness between concurrent TCP connections is remarkably improved if the EFCM controller is used.

Table 9.3: Simulation results of simulation 1, scenario 1 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	39.97	38.86	$-$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	38.52	41.39	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7916	0.8316	$+$

9.3.2 Unreliable Last Hop

Table 9.4 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, the EFCM controller achieves a huge performance gain of approximately 68 % for the overall mean throughput but a slight performance degradation of approximately 1 % for the connection-oriented mean throughput.

Table 9.4: Simulation results of simulation 1, scenario 2 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	23.56	41.37	+
	TCP 2	24.20	40.60	+
	TCP 3	24.99	40.20	+
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	69.12	69.35	=
	TCP 2	69.69	67.98	−
	TCP 3	69.78	68.33	−
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	12.33	
	TCP 2	3.00	12.07	
	TCP 3	3.00	12.07	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.64	
	TCP 2	44.00	40.60	
	TCP 3	44.00	40.44	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.046	
	TCP 2	N/A	0.045	
	TCP 3	N/A	0.045	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.033	
	TCP 2	N/A	0.032	
	TCP 3	N/A	0.032	

Table 9.5 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the overall mean throughput is largely increased by approximately 25 % by the EFCM controller. For the connection-oriented mean throughput a slight performance degradation of approximately 4 % can be observed if the EFCM controller is used. But the fairness is largely increased by the EFCM controller.

Table 9.5: Simulation results of simulation 1, scenario 2 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	35.48	44.49	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	65.65	63.23	$-$
$\overline{I}_f$ of concurrent TCP connections []	0.8261	0.8636	$+$

9.4 Simulation 2: TCP NewReno, RTT $\geq$ 60 ms

9.4.1 Reliable Last Hop

Table 9.6 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, the EFCM controller slightly increases the overall mean throughput of new TCP connections by approximately 3 % (not significant). But the connection-oriented mean throughput of these TCP connections is much more increased by approximately 16 % if the EFCM controller is used.

Table 9.6: Simulation results of simulation 2, scenario 1 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	33.82	34.95	$=$
	TCP 2	33.98	34.93	$=$
	TCP 3	33.35	34.15	$=$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	37.12	43.35	$+$
	TCP 2	37.49	43.46	$+$
	TCP 3	37.42	43.18	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	34.56	
	TCP 2	3.00	34.09	
	TCP 3	3.00	34.96	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	70.74	
	TCP 2	44.00	70.08	
	TCP 3	44.00	71.22	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.161	
	TCP 2	N/A	0.159	
	TCP 3	N/A	0.159	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.083	
	TCP 2	N/A	0.081	
	TCP 3	N/A	0.080	

Table 9.7 shows the simulation results for concurrent TCP connections of an ensemble. For these connections, standard TCP and the EFCM controller reach nearly the same overall mean throughput. But the connection-oriented mean throughput is increased by approximately 10 % if the EFCM controller is used. Also the fairness between concurrent TCP connections is remarkably improved if the EFCM controller is performed in the sending end system.

Table 9.7: Simulation results of simulation 2, scenario 1 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	31.93	31.86	$=$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	31.12	34.17	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7943	0.8324	$+$

9.4.2 Unreliable Last Hop

Table 9.8 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, new TCP connections benefit from the EFCM controller with a huge gain of approximately 74 % for the overall mean throughput and a gain of approximately 16 % for the connection-oriented mean throughput.

Table 9.8: Simulation results of simulation 2, scenario 2 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	19.67	34.76	$+$
	TCP 2	20.71	34.52	$+$
	TCP 3	19.75	35.16	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	43.10	49.99	$+$
	TCP 2	43.36	49.81	$+$
	TCP 3	43.09	50.71	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	11.87	
	TCP 2	3.00	12.05	
	TCP 3	3.00	11.98	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.08	
	TCP 2	44.00	40.11	
	TCP 3	44.00	40.34	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.075	
	TCP 2	N/A	0.076	
	TCP 3	N/A	0.075	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.030	
	TCP 2	N/A	0.030	
	TCP 3	N/A	0.031	

Table 9.9 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller achieves a large gain for the overall mean throughput of approximately 33 % and a gain for the connection-oriented throughput of approximately 13 % for concurrent TCP connections. For these TCP connections also a considerable mean fairness improvement of the EFCM controller can be observed.

Table 9.9: Simulation results of simulation 2, scenario 2 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	27.32	36.38	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	39.65	44.77	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.8342	0.8692	$+$

9.5 Simulation 3: TCP NewReno, RTT $\geq$ 100 ms

9.5.1 Reliable Last Hop

Table 9.10 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, new TCP connections benefit from the EFCM controller with a gain of approximately 14 % for the overall mean throughput and a large gain of approximately 31 % for the connection-oriented mean throughput.

Table 9.10: Simulation results of simulation 3, scenario 1 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	30.45	34.29	$+$
	TCP 2	30.52	34.95	$+$
	TCP 3	30.13	34.24	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	31.11	40.87	$+$
	TCP 2	31.20	41.10	$+$
	TCP 3	31.16	40.73	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	39.29	
	TCP 2	3.00	39.15	
	TCP 3	3.00	38.46	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	76.37	
	TCP 2	44.00	76.09	
	TCP 3	44.00	75.39	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.199	
	TCP 2	N/A	0.197	
	TCP 3	N/A	0.196	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.090	
	TCP 2	N/A	0.087	
	TCP 3	N/A	0.088	

Table 9.11 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller achieves a gain for the overall mean throughput of approximately 9 % and a gain for the connection-oriented throughput of approximately 19 % for concurrent TCP connections. For these TCP connections also a remarkable mean fairness improvement of the EFCM controller can be observed.

Table 9.11: Simulation results of simulation 3, scenario 1 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	27.51	30.07	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	25.97	30.93	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7907	0.8174	$+$

9.5.2 Unreliable Last Hop

Table 9.12 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, new TCP connections benefit from the EFCM controller with a huge gain of approximately 76 % for the overall mean throughput and a large gain of approximately 30 % for the connection-oriented mean throughput.

Table 9.12: Simulation results of simulation 3, scenario 2 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	16.94	29.89	$+$
	TCP 2	16.89	29.96	$+$
	TCP 3	16.96	29.76	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	30.38	39.45	$+$
	TCP 2	30.18	39.13	$+$
	TCP 3	30.41	39.29	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	12.03	
	TCP 2	3.00	12.01	
	TCP 3	3.00	12.03	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.26	
	TCP 2	44.00	40.26	
	TCP 3	44.00	40.26	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.111	
	TCP 2	N/A	0.111	
	TCP 3	N/A	0.112	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.034	
	TCP 2	N/A	0.034	
	TCP 3	N/A	0.034	

Table 9.13 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller achieves a large gain for the overall mean throughput of approximately 40 % and a large gain for the connection-oriented throughput of approximately 23 % for concurrent TCP connections. For these TCP connections also a considerable mean fairness improvement of the EFCM controller can be observed.

Table 9.13: Simulation results of simulation 3, scenario 2 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	21.48	29.90	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	27.48	33.89	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.8402	0.8743	$+$

9.6 Simulation 4: TCP NewReno, RTT $\geq$ 500 ms

9.6.1 Reliable Last Hop

Table 9.14 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, new TCP connections benefit from the EFCM controller with a large gain of approximately 33 % for the overall mean throughput and a huge gain of approximately 62 % for the connection-oriented mean throughput.

Table 9.14: Simulation results of simulation 4, scenario 1 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	13.16	16.98	$+$
	TCP 2	12.97	17.34	$+$
	TCP 3	13.10	16.92	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	12.28	19.74	$+$
	TCP 2	12.14	19.81	$+$
	TCP 3	12.22	19.63	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	50.24	
	TCP 2	3.00	48.91	
	TCP 3	3.00	49.48	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	88.57	
	TCP 2	44.00	87.00	
	TCP 3	44.00	87.88	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.564	
	TCP 2	N/A	0.559	
	TCP 3	N/A	0.559	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.149	
	TCP 2	N/A	0.144	
	TCP 3	N/A	0.144	

Table 9.15 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller achieves a large gain for the overall mean throughput of approximately 21 % and a large gain for the connection-oriented throughput of approximately 37 % for concurrent TCP connections. For these TCP connections also a remarkable mean fairness improvement of the EFCM controller can be observed.

Table 9.15: Simulation results of simulation 4, scenario 1 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	11.65	14.06	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	10.45	14.35	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.7757	0.8097	$+$

9.6.2 Unreliable Last Hop

Table 9.16 shows the simulation results for new TCP connections entering an ensemble. Compared to standard TCP, new TCP connections benefit from the EFCM controller with a huge gain of approximately 72 % for the overall mean throughput and a huge gain of approximately 51 % for the connection-oriented mean throughput.

Table 9.16: Simulation results of simulation 4, scenario 2 (new TCP connections)

		no EFCM	EFCM	Δ
$\overline{TPO}$ of new TCP connections [segments/second]	TCP 1	6.30	10.87	$+$
	TCP 2	6.31	10.82	$+$
	TCP 3	6.29	10.77	$+$
$\overline{TPC}$ of a new TCP connection [segments/second]	TCP 1	7.78	11.82	$+$
	TCP 2	7.80	11.77	$+$
	TCP 3	7.79	11.78	$+$
$\overline{\text{ICWND}}$ of a new TCP connection [segments]	TCP 1	3.00	12.56	
	TCP 2	3.00	12.56	
	TCP 3	3.00	12.61	
$\overline{\text{ISSTHRESH}}$ of a new TCP connection [segments]	TCP 1	44.00	40.51	
	TCP 2	44.00	40.48	
	TCP 3	44.00	40.62	
$\overline{\text{ISRTT}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.501	
	TCP 2	N/A	0.502	
	TCP 3	N/A	0.503	
$\overline{\text{IRTTVAR}}$ of a new TCP connection [seconds]	TCP 1	N/A	0.093	
	TCP 2	N/A	0.093	
	TCP 3	N/A	0.094	

Table 9.17 shows the simulation results for concurrent TCP connections of an ensemble. Compared to standard TCP, the EFCM controller achieves a large gain for the overall mean throughput of approximately 45 % and a large gain for the connection-oriented throughput of approximately 41 % for concurrent TCP connections. For these TCP connections also a considerable mean fairness improvement of the EFCM controller can be observed.

Table 9.17: Simulation results of simulation 4, scenario 2 (concurrent TCP connections)

	no EFCM	EFCM	Δ
$\overline{TPO}$ of concurrent TCP connections [segments/second]	6.25	9.05	$+$
$\overline{TPC}$ of a concurrent TCP connection [segments/second]	6.38	9.00	$+$
$\overline{I}_f$ of concurrent TCP connections []	0.8571	0.8904	$+$

9.7 Summary and Discussion

The following Figures 9.1 to 9.5 show the mean values of the considered performance metrics dependent on the minimum round trip time. In addition, for every mean value its 99 % confidence interval is stated.

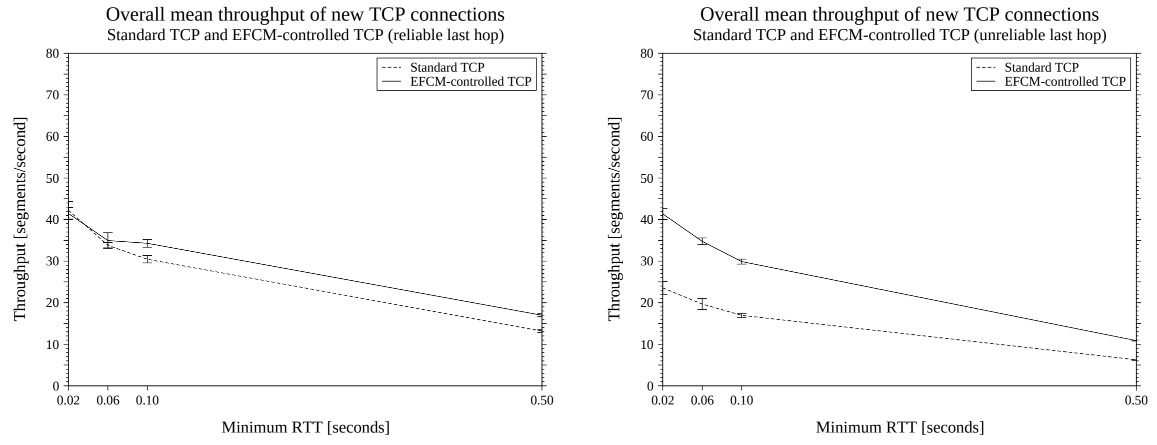


Figure 9.1: Overall mean throughput of new standard or EFCM-controlled TCP connections

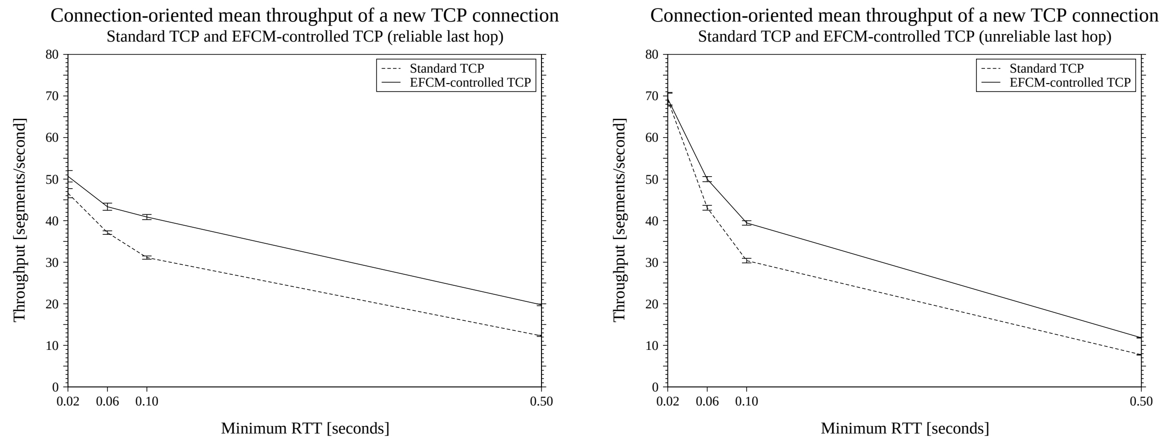


Figure 9.2: Connection-oriented mean throughput of a new standard or EFCM-controlled TCP connection

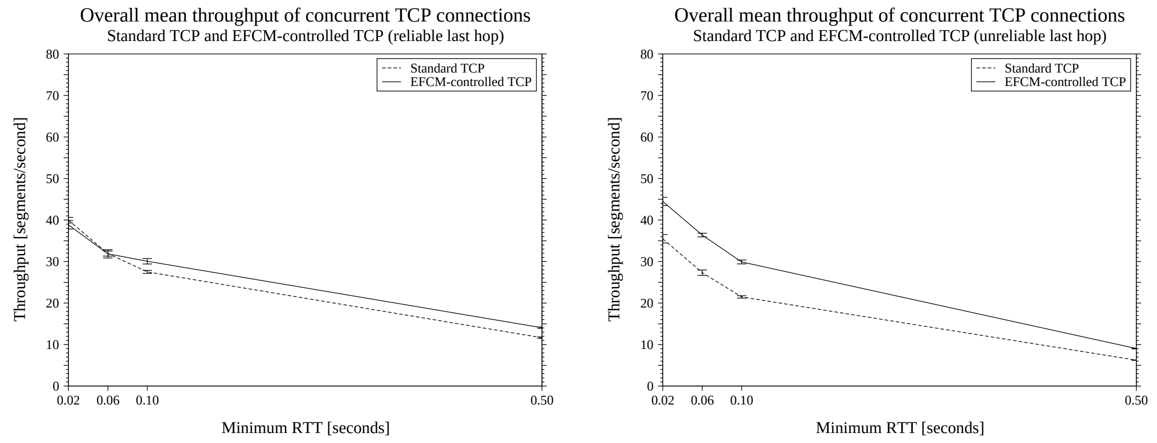


Figure 9.3: Overall mean throughput of concurrent standard or EFCM-controlled TCP connections

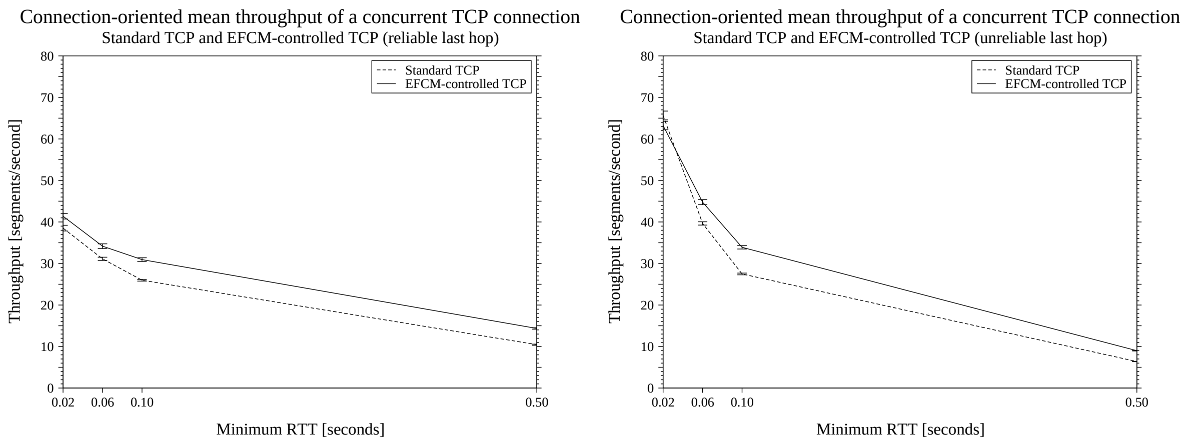


Figure 9.4: Connection-oriented mean throughput of a concurrent standard or EFCM-controlled TCP connection

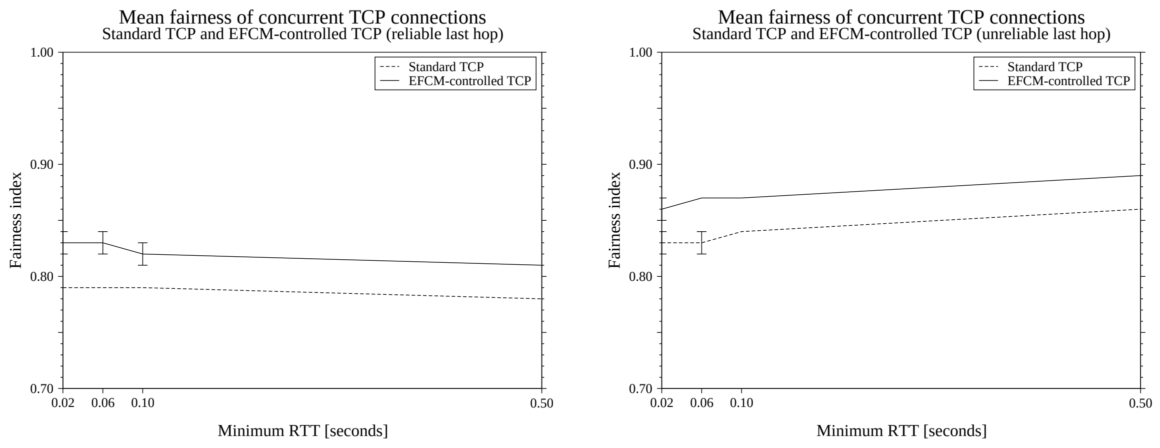


Figure 9.5: Mean fairness index of concurrent standard or EFCM-controlled TCP connections

It can be seen that the benefits and disadvantages of a common congestion control depend to a large degree on the simulation scenario. To summarize:

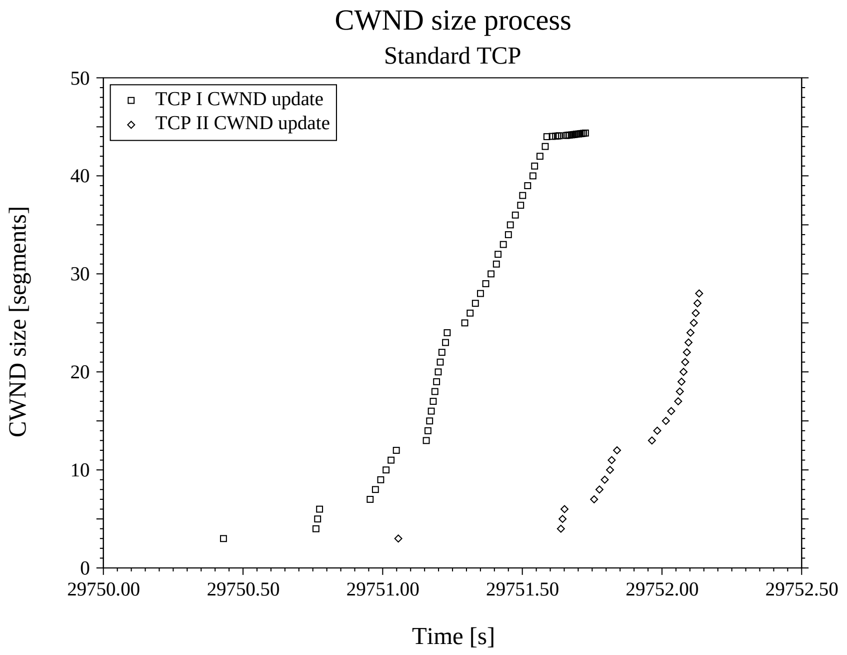
- Reliable last hop scenario: In all considered simulation scenarios the connection-oriented mean throughput of new EFCM-controlled TCP connections is (largely) increased compared to new standard TCP connections. The performance gain of the EFCM controller for this metric is positively correlated with the minimum round trip time and is in the range of 8 % up to 62 %. Thus, a new TCP connection can expect a mean increase in its throughput if the sending end system is equipped with the EFCM controller. If the overall mean throughput of new TCP connections is considered, the EFCM controller reaches a slightly lower and a slightly larger performance in the first two simulation scenarios with minimum round trip times of 20 ms or 60 ms, respectively. But in the simulation scenarios with minimum round trip times above 60 ms, the EFCM controller is able to (largely) improve the overall mean throughput of new TCP connections to gains up to 33 % compared to standard TCP. A similar result can be observed for concurrent TCP connections. EFCM is able to considerably improve the connection-oriented mean throughput of these TCP connections in all simulation scenarios. The performance gain of EFCM for this metric is positively correlated with the minimum round trip time and reaches values between 8 % up to

37 %. For the overall mean throughput of concurrent TCP connections, EFCM slightly decreases this metric in the simulations with lower minimum round trip times less than or equal to 60 ms. The main reason for this behavior of EFCM is in the pacing algorithm which seems to be too conservative for lower minimum round trip times. In Section 5.3.6.3, some possible adaptations of the EFCM pacing algorithm are described that allow a finer control of TCP connections in cases with a lower minimum round trip time. In addition, also the method to calculate the throughput of concurrent TCP connections is important in this context (cf. remarks about this on page 146). For higher minimum round trip times the EFCM controller is able to increase this throughput metric by 9 % up to 21 % compared to standard TCP. In all simulation scenarios, a large fairness gain for concurrent TCP connections can be observed if the EFCM controller is used.

In the considered simulation scenarios with a reliable last hop, the mean initial congestion window is dependent on the minimum round trip time. It increases with the minimum round trip time from 34 up to 49 segments. In the mean, TCP connections with a maximum size of 34 up to 49 segments will largely benefit from the one-time network-information sharing of the EFCM controller, since these TCP connections can transfer all their segments in a single (paced) burst.

Figures 9.6 and 9.7 show some typical processes of the congestion window and the sequence number for either two standard TCP NewReno connections (with a fairness index of $I_f = 0.6098$) or two EFCM-controlled TCP NewReno connections ($I_f = 0.9996$) of an ensemble traversing a network path with a minimum round trip time of 100 ms and a reliable last hop. In Figure 9.6, the concurrent TCP connections are separately controlled and obtain very different congestion windows during their time of concurrency. In Figure 9.7, the concurrent TCP connections are jointly controlled and obtain the same congestion windows during their time of concurrency. And after one of these TCP connections has been closed the remaining TCP connection in the ensemble takes the whole aggregated congestion window as its new congestion window. It can be further seen that the concurrent TCP connections have no bursty sending behavior. This is the result of the built-in pacing mechanism of the EFCM controller. During the slow start phase, some TCP segments are sent sporadically and not in burst of two TCP segments as it is done in standard TCP. This is the outcome of the ensemble ack clocking mechanism used in the EFCM controller.

- Unreliable last hop scenario: In all considered simulation scenarios and for both new TCP connections and concurrent TCP connections, the EFCM controller achieves a large gain in the overall mean throughput compared to standard TCP. New TCP connection entering an ensemble benefit from the EFCM controller with a gain for this throughput metric in the range from 68 % up to 76 %; for concurrent TCP connections the gain for this metric achieved by the EFCM controller is between 25 % and 45 %. Therefore, a common congestion control for TCP connections can partly compensate for the negative influence of packet losses in the last hop of the network on the throughput of single TCP connections. The fairness gain of the EFCM controller for concurrent TCP connections is comparable to the results obtained in the simulations with a reliable last hop. These rather unexpected fairness results can be explained as follows: If two or more TCP connections of one ensemble are established in parallel during one observation period for the fair-



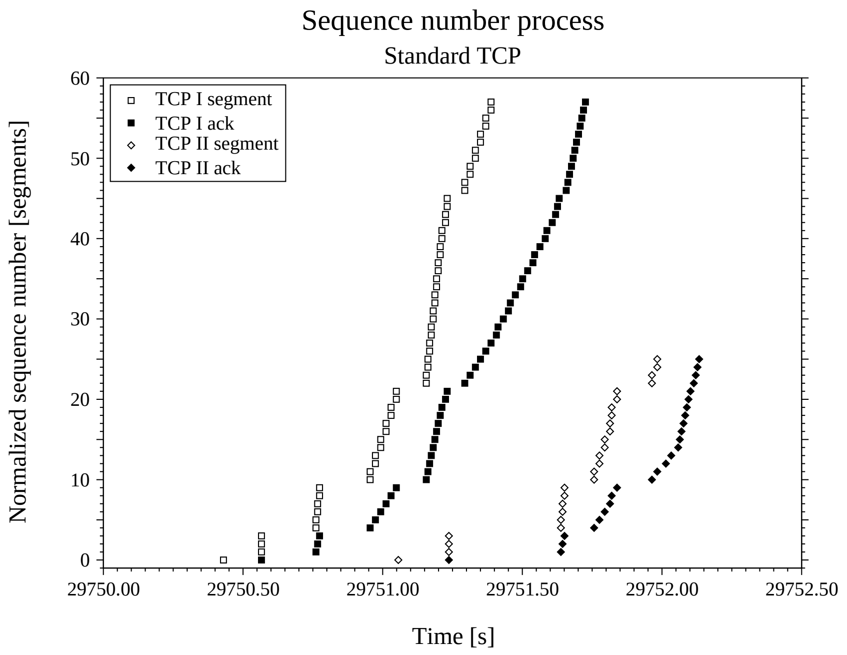
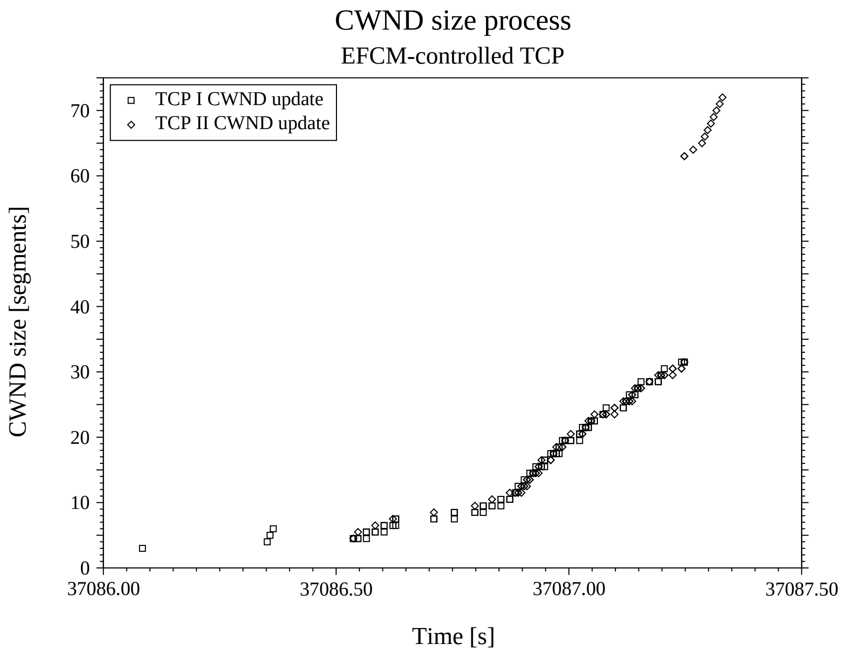


Figure 9.6: Standard TCP connections over a reliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble, respectively

ness index, they have equal values for their jointly controlled TCP variables. This is ensured by the EFCM controller. But the EFCM controller cannot prevent that these TCP connections observe different segment loss patterns during this observation period. Therefore, some TCP senders have to wait for acknowledgments of outstanding segments while other TCP senders can send their (new) segments. As a result, the concurrent TCP connections of an ensemble can have (very) different throughputs in an observation period. Hence, the fairness gain obtained only by the basic EFCM algorithms (cf. Section 5.3.6.2) can be lower. But the ensemble ack clocking mechanism of the



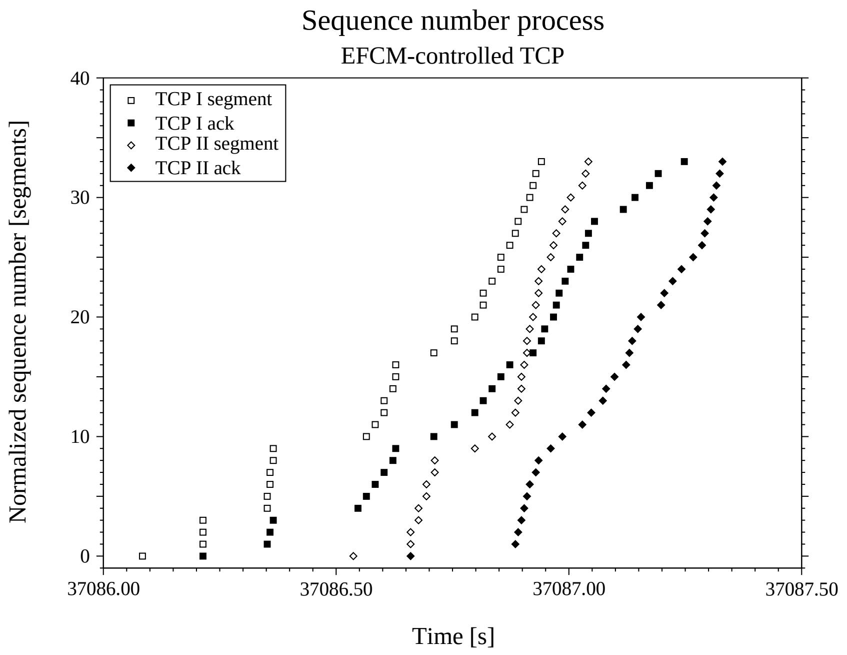
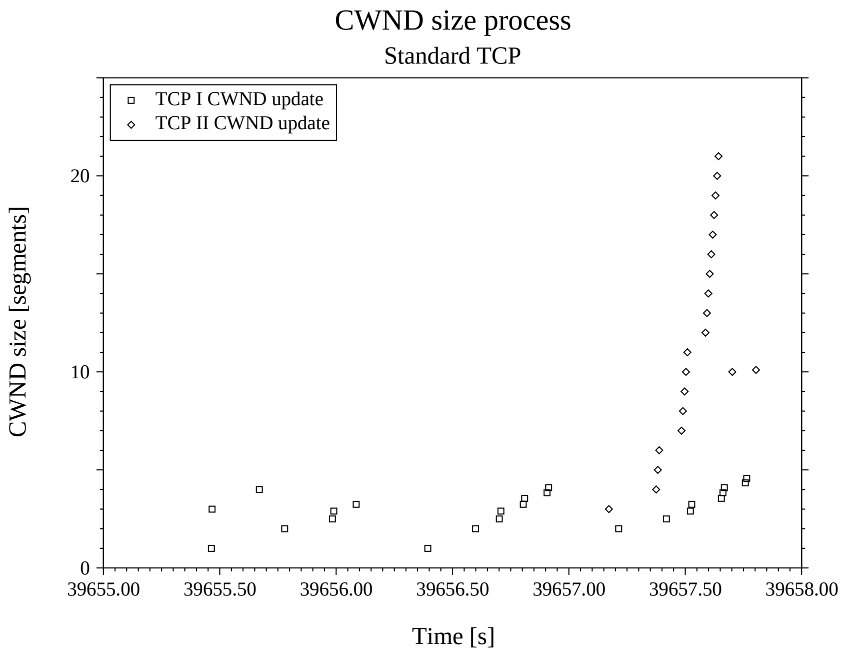


Figure 9.7: EFCM-controlled TCP connections over a reliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble, respectively

EFCM controller is able to eliminate this negative effect, since TCP connections which are waiting for TCP acknowledgments are allowed to send new TCP segments according to their interim increased congestion window. For new TCP connections, the gain for the overall mean throughput is nearly constant if the minimum round trip time increases (the values are in the range of 68 % to 76 %). But the gain for the connection-oriented mean throughput increases with the minimum round trip time. Also concurrent TCP connections reach a larger gain (or a gain at all) for both throughput metrics if the minimum round trip time increases. Thus, the EFCM controller is able



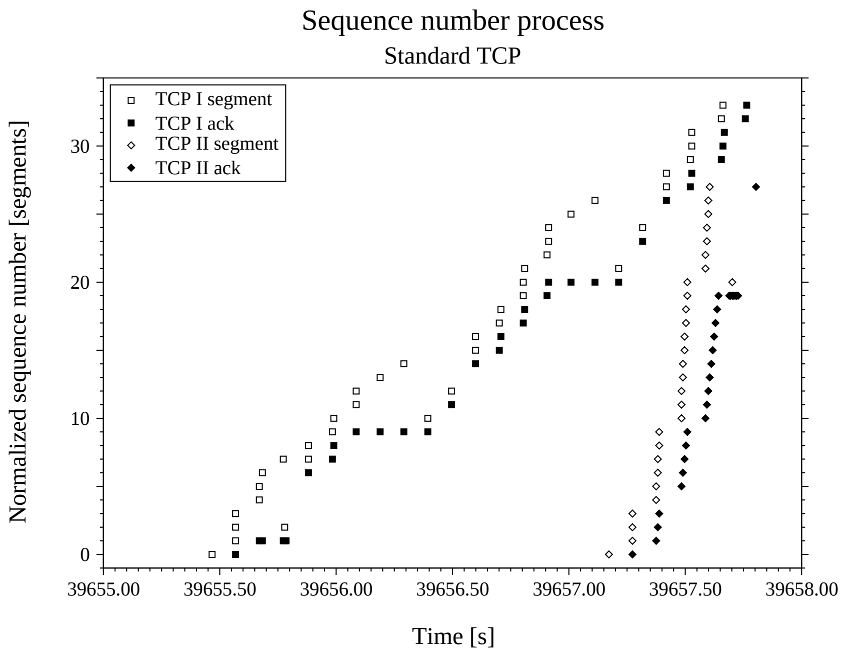
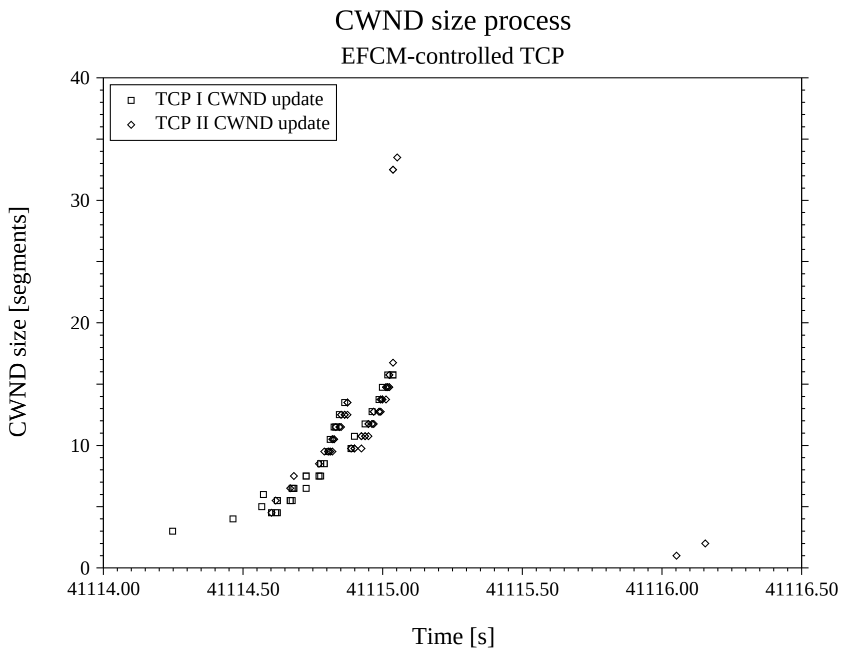


Figure 9.8: Standard TCP connections over an unreliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble, respectively

to compensate the negative influence of sporadic packet losses on the throughput of TCP connections by sharing the decreased congestion window and slow start threshold of the TCP connection affected by packet losses among all TCP connections in an ensemble.

In the considered simulation scenarios with an unreliable last hop, the mean initial congestion window is nearly independent of the minimum round trip time. In the mean, TCP connections with a maximum size of 12 segments will largely benefit from the one-time network-information sharing of the EFCM controller, since these TCP connections can transfer all their segments in a



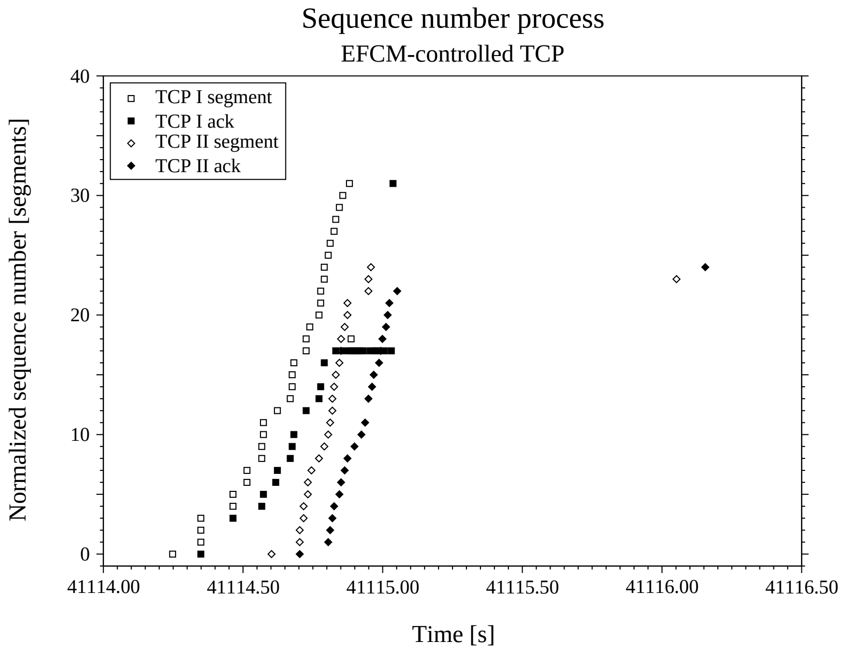
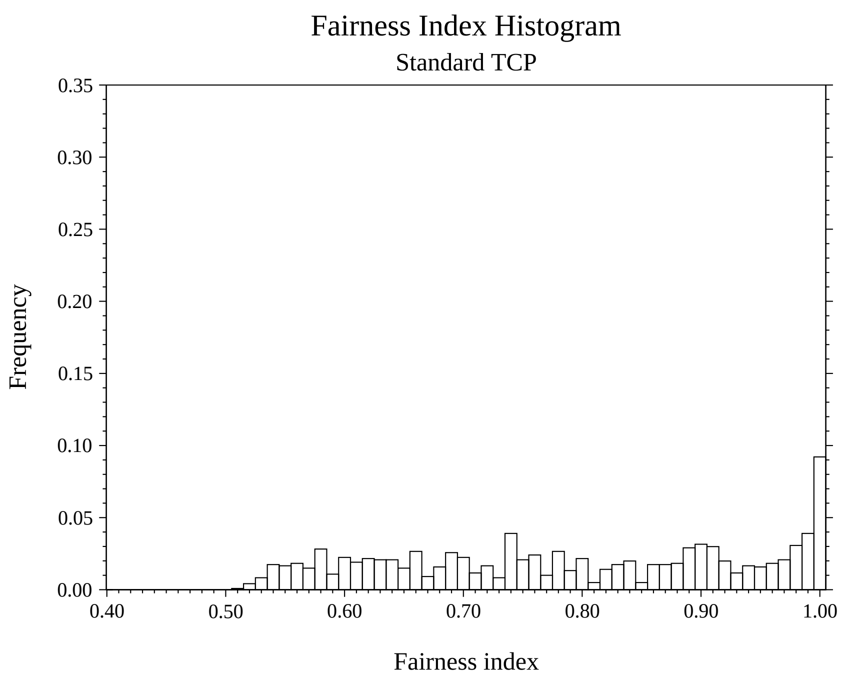


Figure 9.9: EFCM-controlled TCP connections over an unreliable last hop — □ and ◇ represent the first and second TCP connection of an ensemble, respectively

single (paced) burst.

Figures 9.8 and 9.9 show some typical processes of the congestion window and the sequence number for either two standard TCP NewReno connections ($I_f = 0.8118$) or two EFCM-controlled TCP NewReno connections ($I_f = 0.9977$) of an ensemble traversing a network path with a minimum round trip time of 100 ms and an unreliable last hop. In Figure 9.8, the concurrent TCP connections are separately controlled and use different congestion windows during their time of concurrency. In Figure 9.9, the concurrent TCP connections are jointly controlled and use the



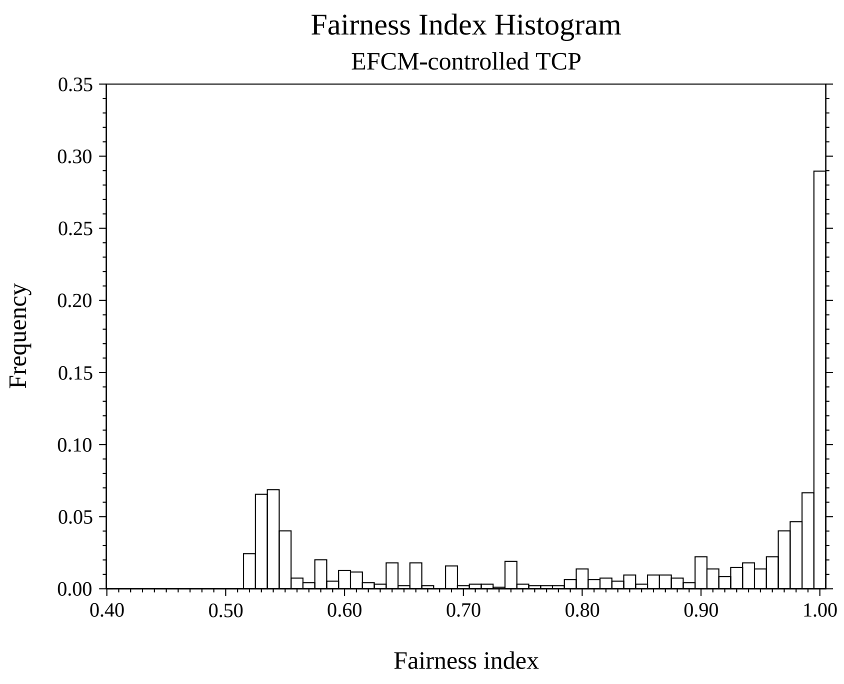
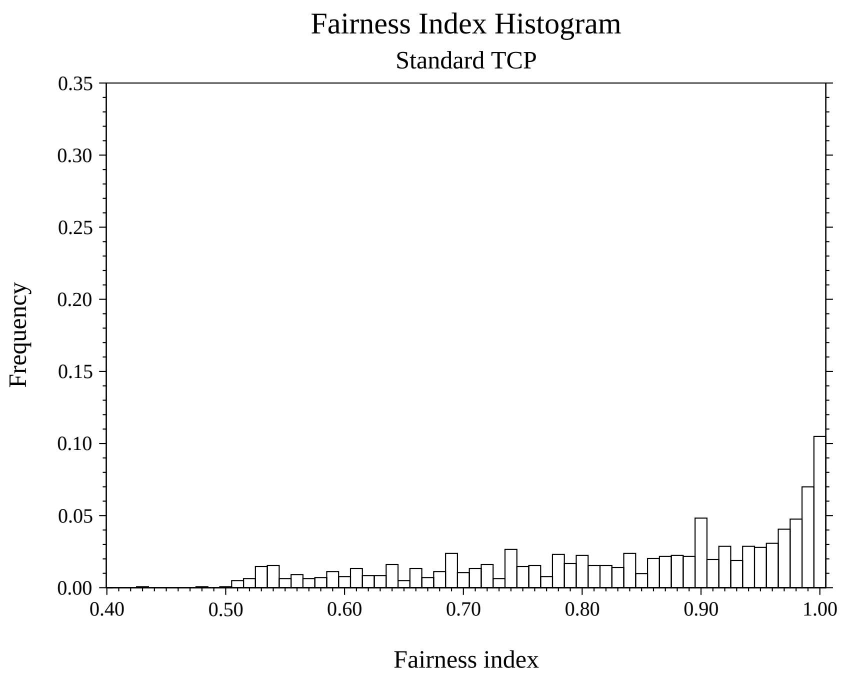


Figure 9.10: Fairness index histogram for concurrent standard or EFCM-controlled TCP connections over a reliable last hop

same congestion windows during their time of concurrency. Due to packet losses some concurrent TCP connections have to wait for a longer period of time to send some new segments. But the negative influence of packet losses on the throughput of TCP connections is noticeably absorbed by a common congestion control.

In the first two simulations with an unreliable last hop the connection-oriented mean throughput (not the overall mean throughput!) of some TCP connections which are controlled or could be controlled by



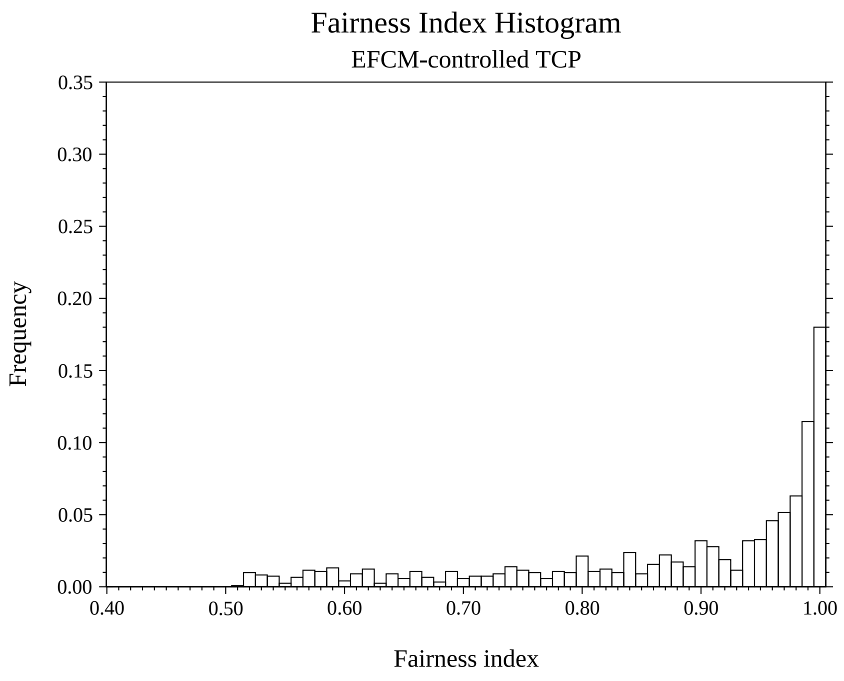
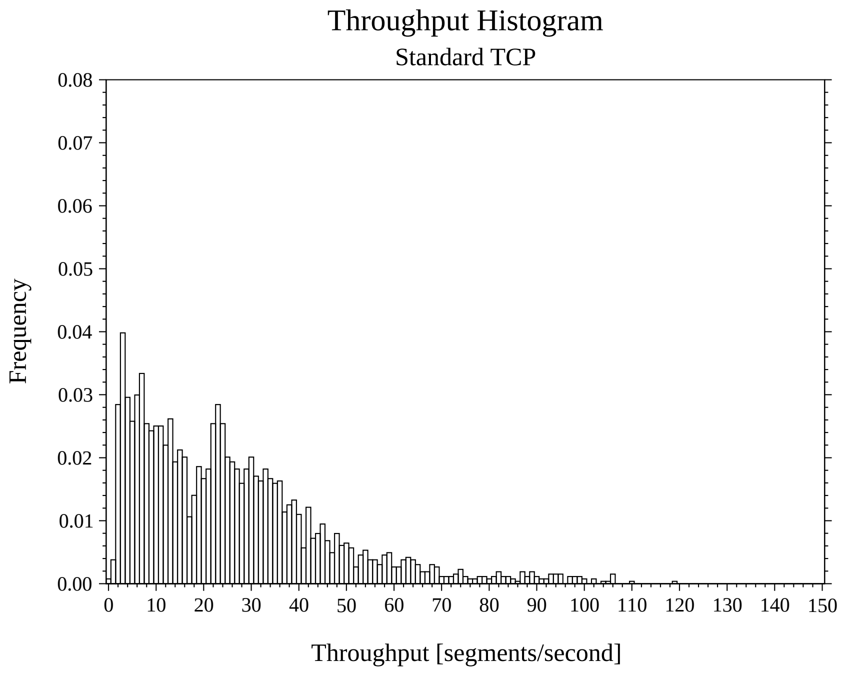


Figure 9.11: Fairness index histogram for concurrent standard or EFCM-controlled TCP connections over an unreliable last hop

the EFCM approach is higher than in the simulation scenarios with a reliable last hop. The same observation has been made in the simulations of the last chapter. An explanation for this at first astonishing result can be found there on page 150.

The throughput results in both last hop scenarios can be explained by taking two effects into account. First, the detrimental influence of TCP's slow start mechanism on the overall performance of a TCP connection increases with the minimum round trip time. Thus, it can be expected that the relative



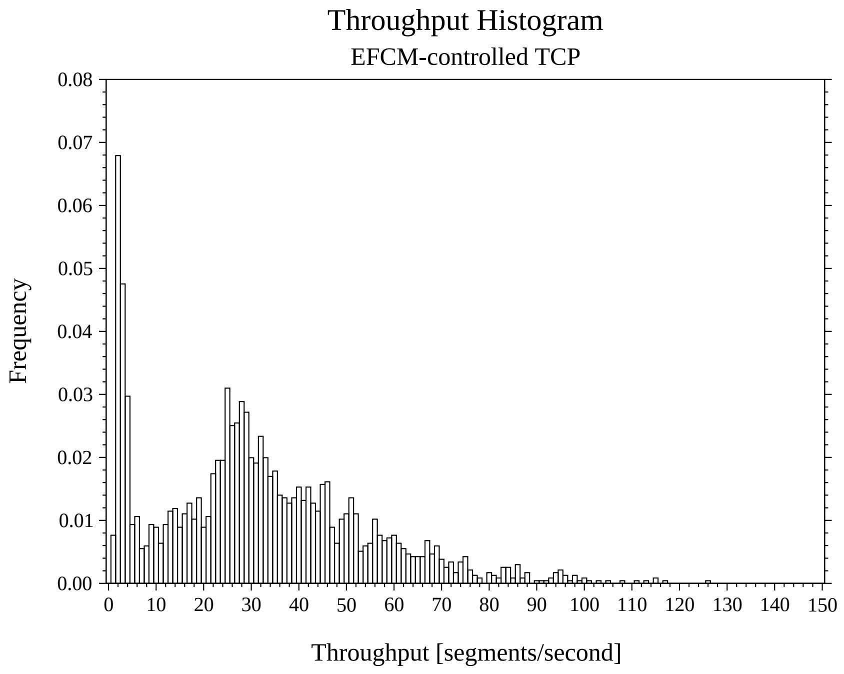
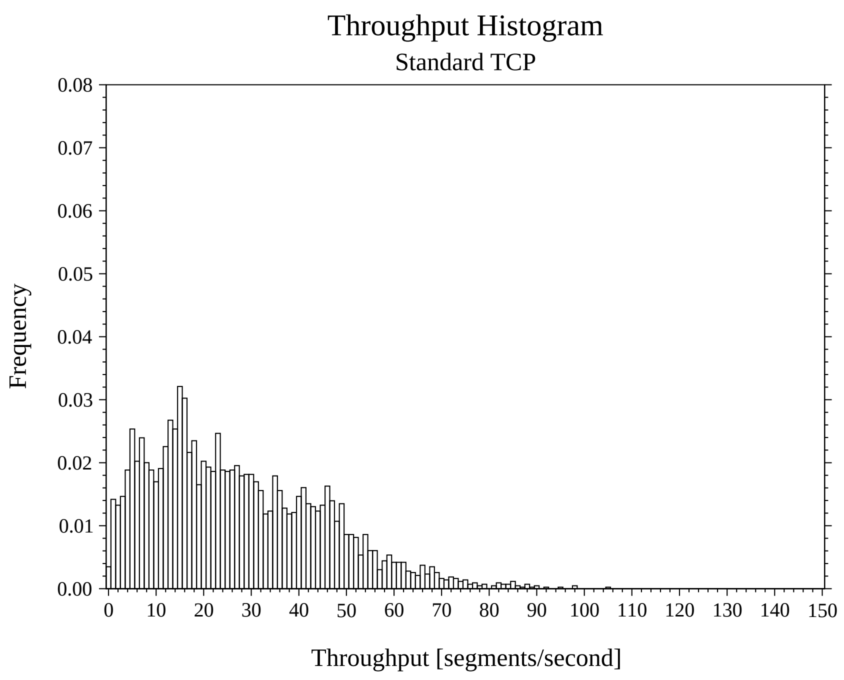


Figure 9.12: Throughput histogram for concurrent standard or EFCM-controlled TCP connections over a reliable last hop

performance gain of the EFCM controller compared to standard TCP also increases with the minimum round trip time. Second, the basic pacing algorithm of the EFCM controller with a constant value of 2 for the parameter α might be too conservative for ensembles with lower minimum round trip times. Thus, it can be expected that the EFCM controller is not able to reach larger throughputs than standard TCP in scenarios with lower minimum round trip times. In these cases, the parameter α of the pacing algorithm should be decreased (cf. Section 5.3.6.3).



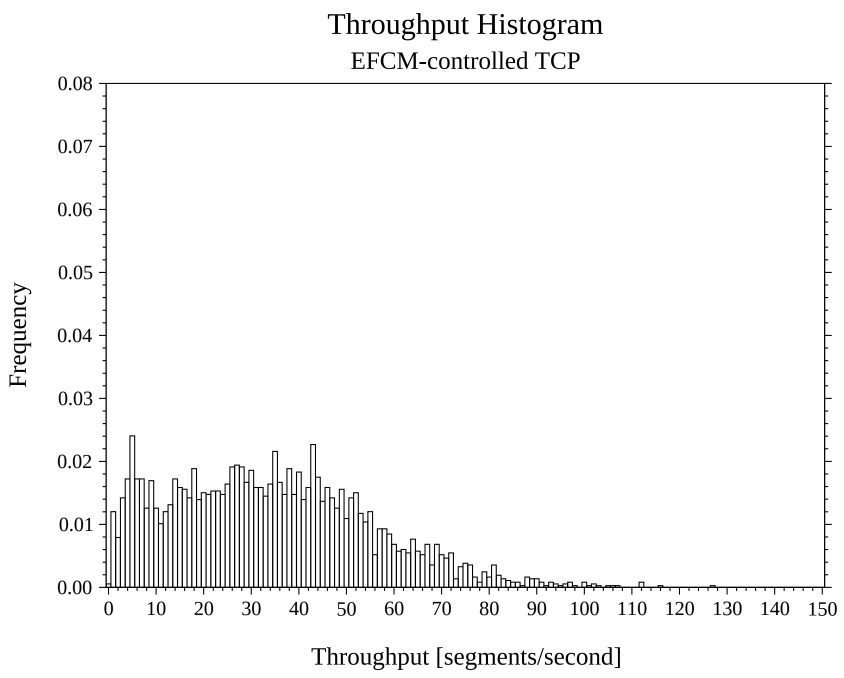


Figure 9.13: Throughput histogram for concurrent standard or EFCM-controlled TCP connections over an unreliable last hop

Figures 9.10 and 9.11 show the histograms of the fairness index for concurrent standard or EFCM-controlled TCP connections for both last hop scenarios in the simulations with a minimum round trip time of 100 ms. Especially for the reliable last hop scenario the positive influence of the EFCM controller on the fairness of concurrent TCP connections is obvious.

Figures 9.12 and 9.13 show the histograms of the throughput of concurrent standard or EFCM-controlled TCP connections for both last hop scenarios in the simulations with a minimum round trip time of 100 ms. The EFCM controller is able to reach rather mean throughputs for the concurrent TCP connections instead of lower or higher throughputs as it is done by standard TCP. The high peaks in Figure 9.12 for very low throughputs are effects of the throughput-calculation method for concurrent TCP connections (cf. Section 7.2.1).

In Table 9.18, the performance of the EFCM controller derived from the simulation results is summarized. A ’+++’, ’++’, or ’+’ denotes that the EFCM controller achieves a huge gain ($\geq$ 50 %), a large gain ($\geq$ 20 %), or a gain ($<$ 20 %) compared to the standard TCP controller and with respect to the performance metric shown in this row; a ’=’ denotes that no significant difference between the standard TCP and the EFCM controller can be observed; and a ’-’ denotes that standard TCP achieves a slight gain compared to the EFCM controller.

Table 9.18: Performance of the EFCM controller compared to standard TCP — $\overline{TPO}_{\text{new}}$ is the overall mean throughput of new TCP connections, $\overline{TPC}_{\text{new}}$ is the connection-oriented mean throughput of new TCP connections, $\overline{TPO}_{\text{concurrent}}$ is the overall mean throughput of concurrent TCP connections, $\overline{TPC}_{\text{concurrent}}$ is the connection-oriented mean throughput of concurrent TCP connections, and $\overline{I}_{f,\text{concurrent}}$ is the mean fairness index for concurrent TCP connections (* = results differ for each of the considered minimum round trip times 20 ms, 60 ms, 100 ms, and 500 ms)

	Reliable last hop	Unreliable last hop
$\overline{TPO}_{\text{new}}$	-,=,+,++*	+++
$\overline{TPC}_{\text{new}}$	+,+,++,+++*	-,+,++,+++*
$\overline{TPO}_{\text{concurrent}}$	-,=,+,++*	++
$\overline{TPC}_{\text{concurrent}}$	+,+,+,++*	-,+,++,++*
$\overline{I}_{f,\text{concurrent}}$	+	+

9.8 Conclusion and Outlook

As expected from the analysis of the EFCM control algorithms [77, 78] (cf. Chapter 6), the simulation results show that the common congestion-control approach EFCM improves the performance of standard TCP under several network conditions. Solely for scenarios with the lowest considered minimum round trip time the EFCM controller reaches lower values for some throughput metrics. For all other scenarios the EFCM controller is able to outperform standard TCP. If, for example, the EFCM controller is used in a scenario with a reliable last hop, a remarkably improved fairness can be observed between concurrent TCP connections of an ensemble. Also the throughput of new and concurrent TCP connections is increased. In the unreliable last hop scenario, EFCM significantly improves the throughput of new and concurrent TCP connections and the fairness between concurrent TCP connections of an ensemble. Particularly for such unreliable last hop scenarios and for TCP connections traversing a network path with a not too low round trip time the usage of EFCM is highly recommended.

These benefits are achieved with relatively simple but well-chosen algorithms performed in the

EFCM controller. Other and more complex algorithms are conceivable, e.g., where the calculation for one TCP control variable depends on the value of another control variable. In addition, if the sender of one TCP connection of an ensemble does not use its whole fair share of the aggregated congestion window in a given time interval then the senders of other TCP connections of the ensemble should increase their congestion window beyond the fair share to make use of the whole aggregated congestion window. The intermittently silent sender of a TCP connection should get a credit for this temporarily unused congestion window to be allowed to have a congestion window higher than the fair share in the future. This new controller mechanism can be implemented by using a utilization factor for each TCP connection in an ensemble. More generally, allowing the controller to adapt congestion windows within an ensemble enables new forms of application support. As an example, the transport layer could better support interactive applications by temporarily assigning their TCP connections a larger share of an ensemble's total bandwidth budget and reducing this share again when the application has no data to send (by compensating TCP connections of other applications for their interim backlog). This can be achieved by using a priority factor for each TCP connection in an ensemble. Since the transport layer does not know which TCP connection belongs to an interactive application, the priority factor of each new TCP connection entering an ensemble should be initially set to a larger value and dynamically decreased to a fixed lower value. Then, every new TCP connection will start with a larger share of the available bandwidth which is continuously decreased to the fair share. But short TCP connections with only a few segments to send will be closed before their priority factor will be (largely) decreased. As the desired result, these short TCP connections reach a larger throughput than longer TCP connections. Such an extended EFCM controller will be able to provide a controlled unfairness among TCP connections of an ensemble and will increase the user's contentment using such an equipped sending end system. But in this case, the effect of such a controlled unfairness on the overall throughput of an ensemble has to be carefully studied.

Another important enhancement of the current EFCM controller is that the scope of an ensemble can be extended by including information from recently closed TCP connections, assuming that the network conditions are slowly varying at best; the speed with which such information becomes inaccurate is an important parameter for such an undertaking. In a similar vein, the EFCM approach could also be used for handover TCP connections, i.e., TCP connections whose receivers are located in mobile receiving end systems which perform a handover. In this case, the handover must be non-transparent for the transport layer of the EFCM-capable sending end system.

Chapter 10

Performance Evaluation of EFCM by Measurements

10.1 Introduction

In Chapter 9, the performance of the EFCM controller has been investigated in a ns-2 simulation framework. The next step is to evaluate the performance of the EFCM controller by measurements in the real Internet environment. As a case-study, the operating system Linux (kernel version 2.4.10) has been selected to implement the EFCM control algorithms in an Internet end system. In this chapter, the original Linux-TCP implementation and the implementation of the Linux EFCM controller are briefly depicted and explained with regard to the standardized TCP algorithms described in Section 2.3.2 and the original EFCM controller illustrated in Section 5.3. Also the measurement setup is conceptually described and the main performance results of the performed measurements are presented. Much more details of the Linux implementation of the EFCM controller, the measurement setup, and the performance results can be found in [82].

10.2 Linux TCP

The basis for implementing EFCM in Linux is the Linux TCP implementation. Linux supports two different TCP variants: TCP NewReno [11] and TCP SACK [31]. These two TCP variants can be combined with different standardized or experimental TCP extensions. Examples are an extended TCP SACK option [83], the TCP timestamp option [84] to allow more frequently and in general more accurate RTT measurements, TCP-Eifel [85] to make TCP's congestion control more robust in case of spurious retransmissions (combined with a new RTO-calculation [34] spurious retransmissions can be reduced or even prevented), and ECN [15]. An overview about all the features and specialties of Linux TCP can be found in [86].

Linux EFCM is implemented on the basis of the Linux TCP NewReno implementation without any TCP extensions. This Linux TCP NewReno implementation mainly follows the standardized TCP and TCP NewReno mechanisms and algorithms (cf. Section 2.3.2) but diverges from them in some specific parts:

(1) The TCP control variables CWND and SSTHRESH of Linux TCP NewReno are measured in segments of maximum size and not in bytes.

(2) Linux TCP NewReno dynamically sets the threshold for activating fast retransmit. Thus, the third duplicate acknowledgment does not trigger fast retransmit in all cases.

(3) If the fast recovery procedure is entered, Linux TCP NewReno does not reduce the CWND in a single step. The CWND is gradually decreased by one segment after every reception of two acknowledgments, until the CWND has reached half of its original value. This new mechanism was originally intended in [87] and later named Rate-halving. Another difference is during fast recovery where Linux TCP NewReno keeps the CWND and does not artificially inflate it.

(4) Linux TCP NewReno uses an adapted calculation of the RTTVAR in cases of largely decreased RTTs:

$$\text{RTTVAR}(T_k) = \begin{cases} (1-\beta')\cdot\text{RTTVAR}(T_{k-1})+ & \\ \beta'\cdot|\text{SRTT}(T_k)-\text{RTT}(T_k)| & \text{if RTT}(T_k) < \text{SRTT}(T_k) \text{ and} \\ & \text{RTTVAR}(T_{k-1}) < |\text{SRTT}(T_k)-\text{RTT}(T_k)| \\ (1-\beta)\cdot\text{RTTVAR}(T_{k-1})+ & \\ \beta\cdot|\text{SRTT}(T_k)-\text{RTT}(T_k)| & \text{else} \end{cases} \tag{10.1}$$

with $\beta' = 1/32$ and $\beta = 1/4$ (standard values).

(5) Also the RTO in Linux TCP NewReno is differently calculated by using a lower limit for the RTO of $G = 200$ ms (cf. Equation (2.5)).

Although Linux TCP differs in some cases from the TCP standardized by the Internet Engineering Task Force (IETF) and described in Section 2.3.2, it keeps the essential properties of TCP regarding the congestion control and conservation of packets.

Compared to the ns-2 implementation of TCP NewReno the Linux implementation of TCP NewReno mainly differs in the parts (2) to (5). In part (5), the ns-2 implementation of TCP NewReno uses the standard-conform lower limit for the RTO of $G = 1$ s which results in a longer retransmission delay. The effects of the other differences between ns-2 and Linux cannot be quantified a priori. But these differences have to be kept in mind if the measurement results of Linux EFCM are compared with the simulation results of EFCM.

10.3 Linux EFCM Controller

10.3.1 Overview

Implementing new congestion-control features for TCP in a real operating system like Linux is a difficult task, since the existing TCP implementation of Linux is speed-optimized and therefore not (well-)designed for extensions. That is the reason why the implementation of some EFCM-specific control algorithms, e.g., the pacing and the ensemble ack clocking mechanism, in Linux has been identified as very difficult tasks. In addition, a real operating system has much more restrictions than a simulation platform like ns-2 regarding the existing types of variables, the provided timer granularity, or the allowed delay of instruction-executions to keep the time-critical conditions of a multitasking-capable operating system. All these differences between reality and simulations lead to necessary adaptations of the original EFCM control algorithms described in Section 5.3 to implement EFCM-like functionalities in Linux. In addition, some of the differences between the original EFCM algorithms and the implementation of the EFCM algorithms in Linux are based on new ideas developed in the meantime to further increase the performance of the EFCM controller in some specific cases.

10.3.2 Design Principles of the Linux EFCM Controller

There exist two solutions to implement the EFCM control algorithms in the Linux kernel:

- The EFCM control algorithms can be "hard-coded" into the existing source code of the TCP implementation of Linux, or
- the EFCM control algorithms can be implemented as loadable Linux kernel modules in combination with as few as possible changes in the existing source code of the Linux TCP implementation.

The first solution has some speed-advantages over the second one. But the second solution is much easier to test (the kernel modules can be separately loaded and tested) and can be faster ported to future Linux kernel versions. In addition, with the second solution it will be possible to adapt some of the EFCM control algorithms by only loading different versions of some of the EFCM-related kernel modules. And taking the later performed measurements into account: only the second solution will be able to quickly switch on (off) the EFCM control algorithms by simply (un)loading the EFCM-related kernel modules. Otherwise, the whole kernel has to be replaced by booting the end system. Therefore, it has been decided to implement the EFCM control algorithms in Linux following the second solution.

The intention of the module-based Linux kernel implementation of the EFCM control algorithms is further to hide kernel-specific internals from the developer of a new common congestion controller as far as possible. Then, some of the EFCM control algorithms can be later adapted also from non-Linux kernel experts to evaluate the performance of other common congestion-control algorithms in a real measurement environment.

A schematic description of the Linux EFCM controller including the interface communication (up- and downcalls) between the adapted Linux TCP implementation and the EFCM-related module(s) is shown in Figure 5.2 on page 98 of this dissertation.

10.3.3 Main Differences of the Linux EFCM Controller to the Original EFCM Controller

In the following, three main differences of the Linux-implementation of the EFCM controller compared to the original EFCM controller are described. The first two differences are necessary due to limitations of Linux while the third difference can be considered as an improvement of the original EFCM controller for a very specific case.

(1) The Linux kernel does not support floating point values. Thus, the calculation of the fair share of some TCP control variables cannot be performed as shown in Figure 5.1. Therefore, a new fair-share calculation has to be used for these control variables. As an example, the aggregated congestion window at time T_k is differently shared among the n TCP connections in an ensemble:

$$\forall i : 1 \leq i \leq n : \mathrm{CWND}_i(T_k) = \left\lfloor \frac{\mathrm{CWND\_AGG}(T_k)}{n} \right\rfloor \tag{10.2}$$

The remaining segments

$$R(T_k) = \mathrm{CWND\_AGG}(T_k) - n \cdot \left\lfloor \frac{\mathrm{CWND\_AGG}(T_k)}{n} \right\rfloor \tag{10.3}$$

of the aggregated congestion window are used to increase the current congestion window of $R(T_k)$ from n selected TCP connections in the ensemble by one. This selection is performed in a fair manner that every TCP connection of an ensemble gets the same mean portion of the aggregated congestion window calculated over the time. This is reached by simply cyclically increasing the congestion windows of those TCP connections in an ensemble which currently have the lowest values. Thus, fairness between all TCP connections of an ensemble is furthermore guaranteed by this new EFCM controller version, not at any time but for any short time period.

Note that this new EFCM control algorithm restricts the maximum difference between the congestion windows of any pair of TCP connections of an ensemble by one. This is in contrast to the original EFCM control algorithm where the congestion windows of the jointly controlled TCP connections have the same values at any time, even when the congestion windows have to store fractional values of the unit MSS. But since these fractions of a MSS stored in the congestion windows of the original EFCM controller cannot be used to send additional segments of maximum size, i.e., in these cases the aggregated congestion window of an ensemble cannot be completely used as it is done with the new EFCM control algorithm, the new EFCM control algorithm has an aggressiveness regarding the number of outstanding segments that is sometimes slightly higher and sometimes slightly lower than the aggressiveness of the original EFCM controller but remains the same considered over a longer time period. In addition, the new EFCM controller slightly improves the pacing mechanism of the original EFCM controller, since $R(T_k)$ segments of an ensemble are sent more separated over the time (the original EFCM controller sends these segments included in a burst of n segments after the congestion windows of the n EFCM-controlled TCP connections have exceeded the next larger integer value).

This adaptation of the original EFCM controller is primary done to overcome the limitations of the Linux kernel regarding the missing floating-point values. But as a positive side-effect it slightly increases the (expected) performance gain and slightly improves the pacing property of the original EFCM controller, particularly for ensembles consisting of a larger number of TCP connections.

(2) In Linux and in contrast to the simulation tool ns-2, the timer granularity is finite. If, for example, the built-in hardware timer of the computer is used to measure the time or time intervals in Linux,

the smallest possible timer interval is 122 μs. Larger timer intervals can be only multiples of this value. Thus, the timer used in the pacing algorithm of the Linux EFCM controller can be not as precise as in the ns-2 simulations. This has some negative effects on the performance of EFCM only if in an ensemble of EFCM-controlled TCP connections the aggregated smoothed round trip time is short and the aggregated congestion window is large. Then, the calculated value for the timer used in the pacing mechanism reaches values near or even below the lowest Linux timer interval.

But even if the pacing timer in Linux has the exact calculated value the Linux EFCM controller might be not able to trigger TCP connections to send new segments at the correct (pacing) time. This is the outcome of the missing real-time property of Linux so that tasks have to wait for their next process slot assigned by the task scheduler of Linux. As a result, TCP segments are sent later and possibly in a burst. Therefore, it can be expected that the EFCM controller performs better if it is embedded in an operating system with real-time capabilities.

(3) In some test measurements with a very high load in the network path (in these measurements the packet loss rate due to congestion was about 5 % for original Linux TCP connections, an unrealistic high value for typical Internet paths) the Linux EFCM controller increases the observed packet loss rate by a factor of 2–3. The verisimilar reason why the Linux EFCM controller dramatically increases the load in the network path in this very specific case is explained in the following: Since a lot of retransmission timer timeouts occur during the measurements, it can be assumed that the problem with the over-aggressiveness of Linux EFCM is related to bursts of timeouts observed by original Linux TCP connections. One plausible explanation is that EFCM-controlled TCP connections have (much) less timeouts in this specific scenario due to the shared and therefore more up to date (smoothed) round trip time. Then, EFCM-controlled TCP connections can react on packet losses by preventing the timeout congestion-control mechanism and using the fast retransmit / fast recovery procedure which results in a much higher overall aggressiveness.

To overcome this problem, the original EFCM controller has been adapted to be less aggressive in the case of a very high load in the network. This is reached by adding the following rule to the original EFCM control algorithms: If an EFCM-controlled TCP connection is affected by a timeout of its retransmission timer, it is temporarily removed from the ensemble until the retransmission of the lost segment(s) has been successfully completed. This time interval is called retransmission phase. During such a retransmission phase the TCP connection is separated from the ensemble and is controlled by the original Linux TCP controller. And after the retransmission phase has been completed, the separated TCP connection is reintegrated into the ensemble using the current values of the TCP control variables CWND and SSTHRESH of its sender. This is the third version of the Linux EFCM controller described in [82] and therefore it is called Linux EFCM III to distinguish it from the original Linux EFCM (Linux EFCM I) and other Linux EFCM controllers.

Of course, all EFCM control algorithms need extra-time compared to the original Linux TCP control algorithms to be executed in a real Internet end system. This is also not considered in the ns-2 simulations but is important to keep in mind for the interpretation of the performance-evaluation results of the Linux EFCM controller.

10.4 Performance-Evaluation Metrics

The performance-evaluation metrics selected for these measurements are the throughput and the fairness index of new and concurrent TCP connections. Here, as a simplification the latter metric considers only the sent segments during the phases where concurrent TCP connections exist and not the sent and acknowledged segments as it is done in the EFCM simulations (cf. Section 7.2.2).

10.5 Measurement Setup

In this section, the measurement setup used for the performance evaluation of the Linux EFCM controller is briefly described. This setup consists of different network scenarios and of the traffic model used to generate payload for senders of TCP connections in the sending end system.

10.5.1 Network Scenarios

Three different network scenarios are considered for the measurements:

(i) Lossless network path (single reliable laboratory link),

(ii) loss-affected network path (single unreliable laboratory link), and

(iii) loss-affected network path (real-loaded Internet network path).

The first two network scenarios shown in Figure 10.1 are performed in a laboratory testbed using a single link between two end systems with adjustable delay and packet loss rate. A new kernel module located in the TCP receiver is implemented that is used to tune these path properties.

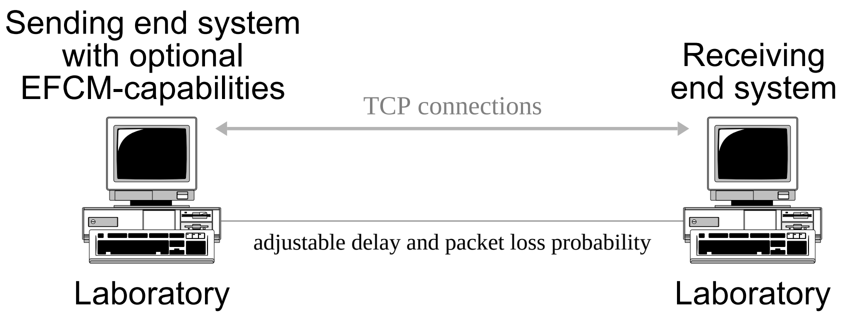

Sending end system with optional EFCM-capabilities
Receiving end system
TCP connections
adjustable delay and packet loss probability
Laboratory
Laboratory

Figure 10.1: Measurement setup for laboratory network scenarios (network scenarios (i) and (ii))

The third network scenario shown in Figure 10.2 is performed using a real Internet network path between an end system located in the TKN environment at the Technical University of Berlin and an end system located at the University of California in Berkeley, USA. For this network path a minimum round trip time of approximately 200 ms has been observed during the measurements.

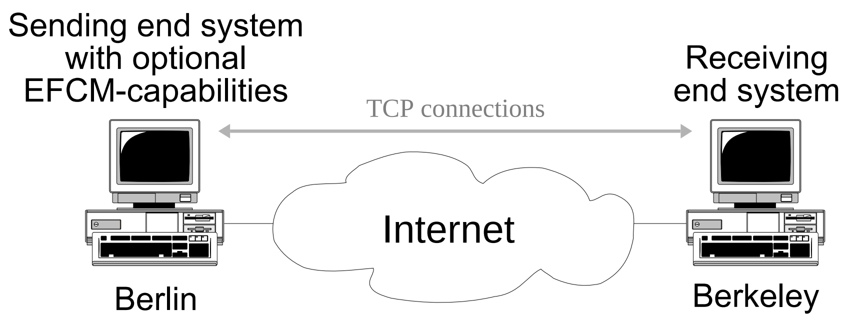


Figure 10.2: Measurement setup for a real network scenario (network scenario (iii))

10.5.2 Traffi c Model

For the measurements a new WWW traffic model derived from the simplified WWW traffic model described in Section 7.3.2 is used. This new WWW traffic model tries to emulate the new HTTP version 1.1 [17] by considering (WWW) pages as the main entity of the model. Such a page consists of a number of WWW objects or files, each of them with a specific file size. Then the overall size of a single page can be calculated by:

$$\text{page size} = \text{number of files} \cdot \text{file size} \qquad (10.4)$$

And such a page is transferred by a single TCP connection as it is allowed in HTTP 1.1. The following Table 10.1 shows the distributions and parameters chosen for the stochastic variables of this new WWW traffic model (cf. Table 7.1).

Table 10.1: Distributions and parameters for the stochastic variables of the new WWW model

Stochastic variable	Distribution	Distribution parameter(s)
Inter-page or Inter-arrival time (IAT)	Exponential	μ is variable e.g. 0.05 s, 0.10 s
Files per page	Lognormal	$\mu = 8.00$ pps $\sigma = 78.752$ pps
File size	Pareto	$\alpha = 1.7584$ $\beta = 30458$ Bytes

The measurements with original Linux TCP and the Linux EFCM controller are performed by starting applications for generating traffic in the sending end system and alternately unloading and loading the EFCM-related kernel modules in the sending end system for a sufficient large measurement interval.

10.6 Main Performance-Evaluation Results

As an example, the main measurement results of the real network scenario (iii) are shown in the following Figure 10.3 (for each mean value the confidence interval for a confidence level of $\alpha = 0.99$ is pictured).

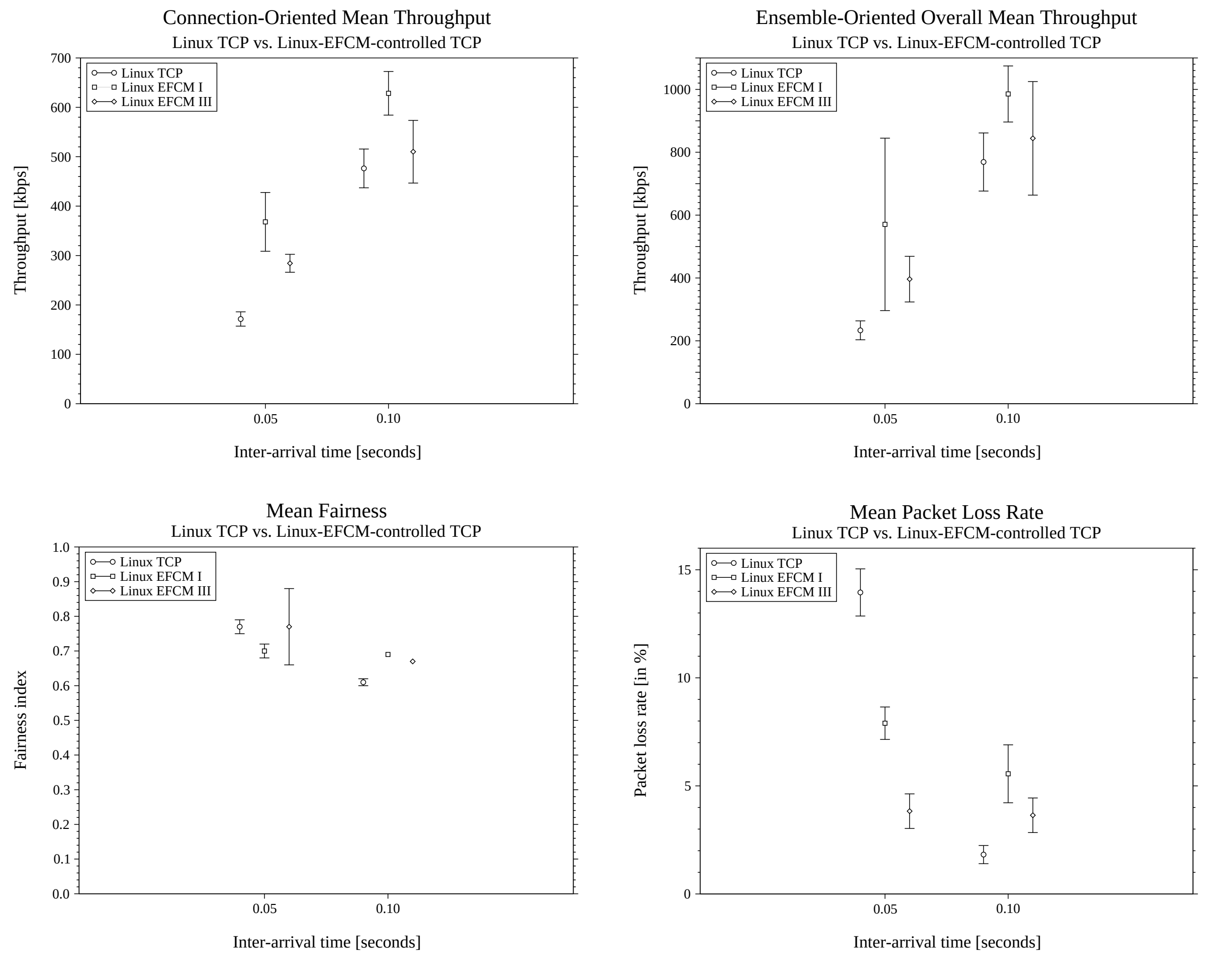


Figure 10.3: Some measurement results of the Linux EFCM controller in the real network scenario

It can be seen that both Linux EFCM variants are able to significantly increase the mean connection-oriented throughput of new TCP connections and the mean throughput of concurrent TCP connections in an ensemble—at least in the case with a much higher load (IAT of 0.05 s). The mean fairness of concurrent TCP connections is only slightly improved by Linux EFCM in the case of a lower load (IAT of 0.10 s). In this case, the original Linux TCP shows some advantages over Linux EFCM regarding the mean packet loss rate. But if the load increases Linux EFCM shows a huge gain compared to Linux TCP for this performance metric. Much more measurement results with regard to the laboratory scenarios and to different versions of the Linux EFCM controller are presented in [82].

In the following, the main results of the performance evaluation of Linux EFCM by measurements are summarized:

- In the network scenario with a lossless network path the Linux EFCM controller reaches a throughput gain of approximately 10 % compared to original Linux TCP. This throughput gain of EFCM is increased to approximately 60 % if a loss-affected network path with an unreliable link is considered. If packet losses in a network path are caused by congestion, the Linux EFCM controller is also able to increase the throughput of the TCP connections in an ensemble. If a network path is highly loaded, this throughput gain of the EFCM without the new algorithm (3) is combined with a largely increased packet loss rate. But if this algorithm (3) is used in the Linux EFCM controller, its throughput gain is largely decreased but still present and the packet loss rate is similar to that observed with original Linux TCP connections. Therefore, algorithm (3) is able to improve the overall performance of the Linux EFCM controller in a very specific network scenario.

- Linux EFCM reaches a 10 % higher mean fairness (index) between concurrent EFCM-controlled TCP connections in an ensemble compared to concurrent separately controlled original Linux TCP connections.

- In general, the Linux EFCM controller performs better compared to the original Linux TCP if the ensemble consists of a larger mean number of TCP connections. For ensembles consisting of less than three TCP connections in the mean, the performance of Linux EFCM and the original Linux TCP are comparable. This is in contrast to the simulation results shown in Chapter 9 where also small ensembles largely benefit from the EFCM controller under certain network path conditions. One possible reason for this measured result is that the control algorithms of the Linux EFCM controller consume more time than expected (the current implementation of the Linux EFCM controller is not speed-optimized and operates on a comparative slow PC with a clock-frequency of 450 MHz) that the existing benefits shown in the simulations are covered by the overhead introduced by the Linux EFCM control algorithms. This might be also the reason why the pacing and the ensemble ack clocking of the Linux EFCM controller show instable behavior if an ensemble consists of more than eighty TCP connections. But since the need for supporting such a large ensemble is rather unlikely, this is not a critical property of the current Linux EFCM controller.

10.7 Conclusion

This chapter has shown that it is possible to implement the EFCM control algorithms and their modifications, respectively, in a real operating system like Linux running on an Internet end system. Measurements have shown that the performance gain of TCP connections controlled by this Linux EFCM implementation and compared to original Linux TCP connections are largely comparable to the simulation results presented in Chapter 9. The differences regarding the minimum number for which the Linux EFCM and the original EFCM controller reach noticeable better performance results than standard (Linux) TCP can be explained by the additional overhead in time introduced by the execution of the EFCM control algorithms in a real (and relative slow) Internet end system.

The modified Linux kernel and all EFCM-related Linux modules are electronically available that Internet end systems equipped with Linux can be easily provided with EFCM.

10.8 General Comparison of Simulations and Measurements

In general, simulations are performed using an abstract model of the reality. Thus, not all aspects of the reality are and can be considered in the simulations. Considering EFCM, for example, the time-consumption of its control algorithms is not considered in the simulations but has an effect on the characteristics of EFCM in the measurements. Depending on the speed of the PC and the capabilities of the operating system in which the EFCM controller is implemented, the EFCM control algorithms require more or less additional time compared to the standard TCP controller. And for Linux this additional time can be only approximated in the mean with a large variability, since Linux is not developed to support real-time network protocols. As a result, using EFCM on the basis of Linux can even interfere some of the EFCM control algorithms. For example, due to the multitasking capabilities of Linux, it cannot be guaranteed that the process performing the pacing mechanism of EFCM is called at the right time. And if the time between consecutive pacing-process calls noticeably differ from the calculated pacing time (cf. Section 5.3.6.3), it can happen that some TCP segments that should be separated over the time are nevertheless sent in a burst. Then, the important feature of EFCM to avoid bursts of TCP segments in the network is violated.

Thus, properties and performance results of congestion-control algorithms observed by simulations have to be carefully verified by measurements in a real network environment—if possible. In general, developers of new congestion-control algorithms should take into account if and how their algorithms can be implemented in real operating systems and which features of their algorithms can or cannot be kept by the specific characteristics of the selected operating system.

Chapter 11

Network-Information Sharing in Internet End Systems: Conclusion and Outlook

The first part of this dissertation has considered NIS approaches for TCP based on one-time or continuous network-information sharing in sending Internet end systems. It has been found out that existing NIS approaches are too limited in their applicability or their (expected) performance gain or are too complex compared to standard TCP. Therefore, two new of such NIS approaches, one called FS-TCBI for one-time and the other one called EFCM for continuous network-information sharing, have been developed whose (expected) performance gain is medium to large and whose additional complexity is low to medium compared to standard TCP. In addition, for EFCM it has been analytically shown that its algorithms regarding the basic fair-share calculation of the CWND and SSTHRESH in an ensemble are optimal, since these algorithms imitate the best-case behavior of standard TCP with (nearly) the same aggressiveness regarding the maximum possible number of outstanding TCP segments of an ensemble compared to standard TCP. Thus, for these two TCP control variables no better network-information sharing algorithms than those used in EFCM are possible. The expected performance gain of EFCM is then based on the fact that standard TCP is not able to keep its best-case behavior for a longer duration if it reaches it at all.

Both new NIS approaches are implemented in a simulation environment to investigate their performance by simulations. Furthermore, EFCM is implemented in Linux running on a real sending Internet end system to show that the complexity of EFCM is controllable and to investigate its performance also by measurements. For FS-TCBI, the simulation results show that a one-time network-information sharing approach is able to improve the throughput of new TCP connections entering an ensemble. But in most cases this is combined with a slightly reduced fairness between concurrent TCP connections of an ensemble and a (slightly) decreased throughput of the whole ensemble. This is the result of the inherent suboptimal behavior of a FS-TCBI-controlled ensemble or of an ensemble controlled by another one-time network-information sharing approach regarding the performance: fairness between existing TCP connections of an ensemble and a new TCP connection entering this ensemble is reached only for the initial point in time where the time of concurrency starts but cannot be guaranteed over the whole time of concurrency. Thus, since the best-case behavior of standard TCP cannot be kept by one-time network-information sharing approaches for a longer duration, the performance of such NIS approaches is far from optimality. Since EFCM is able to imitate TCP's best-case behavior over the whole time of concurrency of TCP connections of an ensemble, its performance gain is much higher than that of FS-TCBI regarding the fairness between TCP connections of an ensemble and the ensemble throughput. This has been shown by simulations and by measurements.

As the main result of the first part of this dissertation, it is recommended to use the EFCM approach in a real sending end system to improve the performance of those TCP connections with TCP receivers in the same receiving end system. In general, for these TCP connections a large performance gain can be assumed if they are EFCM-controlled. But extending this basic common congestion control to a common congestion control for TCP connections with TCP receivers in different receiving end systems of a single subnet is still a practical challenge. The IP addresses of these receiving end systems cannot be used as a decision criterion to decide which TCP connections can form an ensemble, since standardized approaches like Mobile IP, NAT, or even the private installation of subnets of unknown size prevent a valid mapping of IP addresses to the exact location of receiving end systems. Nevertheless, solving this preliminary problem by developing other decision criteria for forming an ensemble is worthwhile, since then the EFCM controller can be used for much more TCP connections of a real sending end system. But the focus of the first part of this dissertation has been on the performance gain such a common congestion control features and not on the difficulties such an approach will have to be used in the real Internet.

A very interesting topic for future research related to common congestion controllers is to use the EFCM algorithms which provide a controlled fairness between TCP connections of an ensemble as a basis for developing algorithms that support a controlled unfairness between TCP connections of an ensemble. Then, as an example, the transport layer could better support interactive applications by temporarily assigning their TCP connections a larger share of an ensemble's total bandwidth budget and reducing this share again when the application has no data to send (by compensating TCP connections of other applications for their interim backlog). This will increase the user's contentment using such an equipped sending end system.

Part II

Network-Information Sharing between Internet Routers and End Systems—Congestion Feedback from Routers

Chapter 12

Congestion Feedback from Routers: Related Work

12.1 Introduction

Two RCF approaches, RED [14] and ECN [15], have been already described. But their performance gain is rather limited (cf. Section 2.3.2.6.4). In order to control the load in the network more accurately, some more complex and more powerful congestion-feedback approaches in routers and end systems than ECN and RED have been developed. The algorithms, mechanisms, and properties of the most important of these NIS approaches based on router congestion feedback (RCF) are described in the following Sections 12.2 to 12.6. The RCF approaches considered in this chapter are: Explicit Window Adaptation (EWA) [88, 89], Explicit Control Protocol (XCP) [90, 91], Core-Stateless Fair Queueing (CSFQ) [92], Fair Bandwidth Allocation for TCP (FBA-TCP) [93], and TCP Quick-Start (QS-TCP) [94].

EWA, CSFQ, and FBA-TCP can be classified as continuous one-way RCF approaches for TCP. While EWA and FBA-TCP use explicit congestion feedback from routers, CSFQ is based on an implicit network-information sharing using packet losses. XCP is a continuous two-way RCF approach based on a new transport protocol with a changed congestion control compared to TCP. Finally, QS-TCP is a one-time two-way RCF approach. It is developed to enable TCP senders to start with larger initial congestion windows according to the feedback information obtained from QS-TCP-capable routers.

In addition to these RCF approaches, two new continuous one-way RCF approaches are considered in detail in the next chapter. The first new approach, Fuzzy Explicit Window Adaptation (FEWA), is based on EWA. It modifies parts of the feedback-calculation algorithms of EWA in the routers. The second new RCF approach, Enhanced TCP (ETCP), uses the feedback information from FEWA-capable routers for an adapted congestion control in the TCP senders.

The properties and functionalities of all these RCF approaches are summarized and compared in Chapter 14.2. Based on this comparison, among these RCF approaches some potential candidates for a network-information sharing in future IP-based networks are selected for a detailed investigation of their impact on the performance of the network.

12.2 Explicit Window Adaptation (EWA)

The EWA approach [88, 89] has been developed to explicitly inform the senders of TCP connections about the currently available bandwidth over a bottleneck link in a transparent way by using TCP's built-in flow-control mechanism. Thus, the source codes of a TCP sender and a TCP receiver are kept unchanged. In this section, the basic algorithm of EWA is described, it is shown how EWA can be deployed in the current Internet environment, some shortcomings of EWA are explained, and some possible improvements of EWA are depicted.

12.2.1 EWA Algorithm

After every measurement interval i with a fixed duration dependent on the bandwidths in the links the EWA-capable router is connected with, e.g., 10 ms, a router with EWA-capabilities measures its current queue length Q_i and computes the current mean queue length $\overline{Q}_i$. Q_i, $\overline{Q}_i$, and the second-last computed mean queue length $\overline{Q}_{i-1}$ are then used to calculate a new sending window for each TCP connection traversing this router:

$$\text{sending window} = \max\{\text{MSS}, \alpha \cdot \log_2(B - Q_i) \cdot \text{MSS}\} \tag{12.1}$$

where B is the maximum queue length in the router (i.e., at the same time at most $B + 1$ packets can be stored and forwarded in the router), MSS is the maximum segment size of all TCP connections traversing the router, and α is a dynamically determined factor whose calculation is later explained. B and Q_i are expressed in number of packets and MSS is expressed in number of bytes. The logarithmic expression in Equation (12.1) is introduced to reflect that TCP connections which are in slow start are able to send twice as much segments in their next round trip time (RTT) interval than now. As a result, the queue length of the router can exponentially grow in the near future. In addition, Equation (12.1) ensures that every TCP connection is always allowed to send at least one TCP segment with a MSS.

The alterable factor α in Equation (12.1) is introduced to better utilize the link if only a few TCP connections are transferring segments over the router. α is updated every 10 milliseconds as follows:

$$\alpha = f(\alpha, \overline{Q}_i) = \begin{cases} \alpha + w_{\text{up}} & \text{if } \overline{Q}_i < \text{threshold}_{\text{low}} \\ \alpha \cdot w_{\text{down}} & \text{if } \overline{Q}_i > \text{threshold}_{\text{high}} \end{cases} \tag{12.2}$$

with

$$\overline{Q}_i = \frac{127}{128} \cdot \overline{Q}_{i-1} + \frac{1}{128} \cdot Q_i \tag{12.3}$$

The initial value for the utilization factor α is set to 1, the parameters w_{up} to additively increase and w_{down} to multiplicatively decrease α are set to $1/8$ and $31/32$, and the low and high thresholds for the mean queue length are set to 20 % and 60 % of the maximum queue length B.

The calculated sending window is transferred to each TCP sender by modifying the advertised receiver window in the TCP acknowledgments. It is only *reduced* by the EWA-capable router if necessary, but never increased to maintain TCP's end-to-end flow control:

$$\text{advertised receiver window} = \min\{\text{sending window}, \text{advertised receiver window}\} \tag{12.4}$$

With this explicit congestion-feedback information, the TCP senders are able to react more adequately to the current load in the router than it is possible with other mechanisms, e.g., ECN or RED.

12.2.2 Applicability of EWA

To deploy EWA in a network it is not necessary to equip all routers in the network with EWA-capabilities. It is sufficient to provide only the bottleneck router(s) with EWA.

EWA requires that the segments and acknowledgments of the considered TCP connections pass through the same (bottleneck) routers. If this necessary condition of symmetrical routing can be guaranteed for all (bottleneck) routers in the path from a TCP sender to its TCP receiver, such an approach can be used for an improved end-to-end congestion control based on a flow control between the (bottleneck) routers and the TCP senders.

In addition, if the TCP window scale option [30] is used it must be implemented by performing the continuous window-scale negotiation between a TCP sender and its TCP receiver as it is described in Section 2.3.2.5. Otherwise, the EWA-capable router is not able to correctly evaluate and set the window-field in TCP acknowledgments, since an EWA-capable router cannot store the possibly negotiated window scale factor of every TCP connection traversing it (no per-flow information are available at the router).

12.2.3 Shortcomings of EWA

In my opinion, the basic EWA algorithm described in [88, 89] does not consider the in general most realistic case that the queue of a router with EWA-capabilities is below the lower threshold for a longer period of time. In this case, the utilization factor α perpetually increases without an upper bound. If, after a while, the formerly lowly loaded router is highly loaded due to a large burst of incoming packets, it takes dozens or hundreds of measurement intervals to decrease the utilization factor α to an appropriate value. In the meantime, the EWA algorithm is not able to prevent the router from congestion and packets get lost. Thus, the history of the load of the router is too heavily weighted in the calculation of the utilization factor α of the EWA algorithm. This drawback of the EWA algorithm can be softened but not completely prevented if an upper bound for the utilization factor α is introduced, if the current queue length Q_i is much higher weighted in Equation (12.3), or if the parameter w_{down} is decreased to a value $\ll 1$. But it can be only prevented if a modified calculation of the utilization factor α with a much faster reaction on the current load in the router is used (cf. Section 13.2).

12.2.4 Improvements of EWA

The congestion feedback function (12.1) of EWA is one example function to calculate the sending window in a router. Other feedback functions are conceivable. These new feedback functions should adopt the well-chosen logarithmic expression from the EWA feedback function, but can adapt or replace the calculation of the utilization factor α. The FEWA algorithm described in Section 13.2 is an example approach for the latter case.

EWA and related approaches can be easily adapted so that symmetrical routing is no longer a necessary condition for its usage. This can be done by introducing a new TCP header option that carries the calculated sending window of the routers in the path from the TCP sender to the TCP receiver. The TCP receiver puts the minimum of this sending window and its currently supported receiver window as the new advertised receiver window in a normal TCP acknowledgment and sends it to the TCP sender. For this variant of EWA the TCP in the receiving end systems have to be slightly changed. Although this new implementation of EWA increases the delay of the EWA control loop between the routers and the TCP senders, it is not certain that this will decrease the overall performance of EWA. A positive effect of this new implementation on the performance of EWA is that the sending window of each EWA-capable router is calculated with a higher probability during the period of time the router has to handle segments from those TCP connections which get the feedback information. This is due to the inherent bursty sending behavior of the window-based congestion control of TCP (and in contrast to rate-based congestion-control mechanisms used in other networks, e.g., in ATM networks [2, 3, 4, 5, 6]). As a result, with the new EWA implementation the correlation between the current sending window and the obtained feedback might be increased from a single TCP sender's point of view. Whether the negative or positive effect of this new EWA implementation dominates the overall performance of EWA cannot be answered a priori. This has to be carefully investigated.

12.3 Explicit Control Protocol (XCP)

The XCP [90, 91] is a new transport protocol related to TCP. Unlike TCP, XCP provides explicit congestion feedback from XCP-capable routers to XCP senders. Therefore, XCP senders are able to more adequately control their sending window to reach an efficient, fair, scalable and stable congestion control in the whole network.

The feedback congestion-control algorithm in an XCP-capable router is divided in two parts: an efficiency and a fairness control algorithm. With this approach, efficiency of and fairness among XCP connections in a router can be separately managed. In this section, the protocol mechanisms and the feedback congestion-control algorithms of XCP are described in detail.

12.3.1 XCP Congestion Feedback Transfer

Each data packet of an XCP connection carries a congestion header (CH) (see Figure 12.1). The first

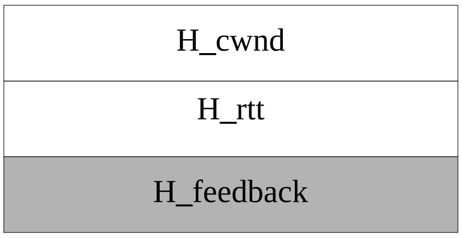


Figure 12.1: Congestion header in an XCP data packet/acknowledgment

two values, H_cwnd and H_rtt, are set by an XCP sender to its current congestion window and its current RTT estimate and are kept unchanged during the transport. The third value H_feedback is used for the congestion feedback of the routers. It is initialized by an XCP sender to the desired increase of its current congestion window and can be modified by a router according to the first two values in the congestion header and the efficiency and fairness control algorithms performed in the router. In more detail: If the XCP sender has a desired sending rate r, the initial value for H_feedback can be computed as follows:

$$\text{H\_feedback} = (r \cdot \text{rtt} - \text{cwnd})/\text{number of packets in congestion window} \tag{12.5}$$

In the first packet of an XCP connections, H_feedback is initialized to 0, since the XCP sender has no valid estimation of the current RTT in the network path.

The XCP receiver copies the congestion header of an arriving data packet to an acknowledgment and sends the acknowledgment including the congestion header back to the XCP sender. After an acknowledgment arrival, the XCP sender adapts its new congestion window according to the router feedback stored in the congestion header:

$$\text{cwnd} = \max\{\text{cwnd} + \text{H\_feedback}, s\} \tag{12.6}$$

where s is the packet size.

The basic XCP sender does not use a pacing mechanism. But in order to avoid a bursty sending behavior of an XCP sender after a large increase of the congestion window, such a pacing mechanism should be additionally implemented in an XCP sender.

12.3.2 XCP Feedback Congestion Controller

As already mentioned, the feedback congestion controller in an XCP-capable router is divided in an efficiency controller (EC) and a fairness controller (FC). The efficiency controller's task is to maximize the link utilization and to minimize the packet loss rate and the persistent queue of a link. The EC deals only with aggregated traffic of a link and does not consider any fairness issues between flows of that aggregated traffic. This is the task of the fairness controller. Using the current per-link congestion feedback information computed by the EC the FC calculates the current per-packet congestion feedback information for each flow. This per-packet congestion information is stored in the H_feedback field of the congestion header in every packet and carried back to each XCP sender.

For each link a router maintains a control timer that is set to the most recent estimate of the average RTT seen by the XCP senders on that link. After every timeout of each link control timer the EC and the FC are used to calculate current values of the feedback congestion control for XCP flows traversing this link.

In the following, the mathematical expressions of the EC and FC calculations are explained. More information about the background of the XCP control algorithms and an example implementation based on a pseudo code can be found in [90, 91].

12.3.2.1 XCP Efficiency Controller (EC)

d is the average RTT estimated for a link, S is the spare bandwidth of a link defined as the difference between the input traffic rate for that link and the link capacity. And Q is the persistent queue size of a link in bytes which is the queue size that does not drain in a round trip propagation delay. Q is computed as the minimum of all queue sizes seen by any arrived packet in the last propagation delay. Then the aggregated congestion feedback ϕ in bytes of this link can be computed as

$$\phi = \alpha \cdot d \cdot S - \beta \cdot Q \tag{12.7}$$

with two constants $\alpha = 0.4$ and $\beta = 0.226$ (cf. [90, 91]). The aggregated congestion feedback ϕ should be proportional to S, since if the link is underutilized ($S > 0$) or congested ($S < 0$) a positive or a negative feedback should be sent to the XCP senders. But ϕ should be also proportional to $-Q$ to drain the queue. For example, if the input traffic rate matches the link capacity, i.e., $S = 0$, the aggregated congestion feedback ϕ must be set to a negative value to drain the queue. Equation (12.7) ensures that the aggregated congestion feedback ϕ of a link is proportional to S and $-Q$.

12.3.2.2 XCP Fairness Controller (FC)

The fairness controller is based on the additive-increase-multiplicative-decrease (AIMD) principle known from TCP, i.e., the per-packet congestion feedback follows this policy:

- If $\phi > 0$, the throughput increase of all flows is the same.
- If $\phi < 0$, the throughput decrease of a flow is proportional to its current throughput.

This policy ensures a continuous fairness convergence if ϕ is not equal to zero. But if the efficiency is approximately optimal, i.e., $\phi \approx 0$, this convergence can be stalled. In order to prevent this, the concept of bandwidth shuffling is used where in every control interval a small amount of traffic h with

$$h = \max\{0, \gamma \cdot y - |\phi|\}, \tag{12.8}$$

with a constant $\gamma = 0.1$ and the input traffic y in an average RTT, is redistributed according to the AIMD principle over all flows.

For a flow i the per-packet congestion feedback H_feedback_i can be written as a linear combination of a positive congestion feedback p_i and a negative congestion feedback n_i:

$$\text{H\_feedback}_i = p_i - n_i \tag{12.9}$$

In a single control interval, p_i and n_i are computed by using the values

$$\xi_p = \frac{h + \max\{\phi, o\}}{d \cdot \sum \frac{\text{H\_rtt}_i \cdot s_i}{\text{H\_cwnd}_i}} \tag{12.10}$$

and

$$\xi_n = \frac{h + \max\{-\phi, o\}}{d \cdot \sum s_i} \tag{12.11}$$

where the sum in each denominator is considered over all packets in a control interval. These two formulas are later explained. Then p_i and n_i can be computed as follows:

$$\begin{aligned} p_i &= \xi_p \cdot \frac{\text{H\_rtt}_i^2 \cdot s_i}{\text{H\_cwnd}_i} && (12.12) \\ n_i &= \xi_n \cdot \text{H\_rtt}_i \cdot s_i && (12.13) \end{aligned}$$

Equation (12.12) can be explained as follows (my own considerations): Due to the policy of the fairness controller, each XCP flow should get the same amount of additional throughput ΔT if the aggregated congestion feedback is above zero, i.e., $\Delta T_i = \Delta T_j = \Delta T$ for any two XCP flows i and j. For each XCP flow i this additional throughput is equal to the additional increase in the congestion window divided by the round trip time of flow i: $\Delta T = \Delta\text{H\_cwnd}_i/\text{H\_rtt}_i$, i.e., $\Delta\text{H\_cwnd}_i = \Delta T \cdot \text{H\_rtt}_i$. This additional increase in the congestion window must be shared over all packets the router sees from the XCP flow i during the control interval d to achieve a per-packet congestion feedback for XCP flow i. During a control interval d $\frac{\text{H\_cwnd}_i}{s_i} \cdot \frac{d}{\text{H\_rtt}_i}$ packets from flow i traverse the router. Therefore, each of these packets should carry a positive congestion feedback of

$$p_i = \frac{\Delta T}{d} \cdot \frac{\text{H\_rtt}_i^2 \cdot s_i}{\text{H\_cwnd}_i} \qquad (12.14)$$

In a control interval additional $h + \max\{\phi, 0\}$ bytes should be allocated by all XCP flows. Thus, the total increase in the aggregated traffic rate is

$$\frac{h + \max\{\phi, 0\}}{d} = \sum \frac{p_i}{\text{H\_rtt}_i} \qquad (12.15)$$

where the sum is considered over all packets during a control interval. If Equation (12.14) is inserted in Equation (12.15) it follows that (cf. Equation (12.10))

$$\Delta T = \frac{h + \max\{\phi, o\}}{\sum \frac{\text{H\_rtt}_i \cdot s_i}{\text{H\_cwnd}_i}} = d \cdot \xi_p \qquad (12.16)$$

Equation (12.13) can be explained as follows (my own considerations): Due to the policy of the fairness controller, each XCP flow i should reduce its future throughput by a throughput-reduction factor ΔR that is proportional to its current throughput if the aggregated congestion feedback ϕ is below zero, i.e., $\Delta R_i = \Delta R_j = \Delta R$ for any two XCP flows i and j. For each XCP flow i this throughput-reduction factor ΔR is equal to the decrease in the congestion window divided by the current congestion window of flow i: $\Delta R = \Delta\text{H\_cwnd}_i/\text{H\_cwnd}_i$, i.e., $\Delta\text{H\_cwnd}_i = \Delta R \cdot \text{H\_cwnd}_i$. This decrease in the congestion window must be shared over all $\frac{\text{H\_cwnd}_i}{s_i} \cdot \frac{d}{\text{H\_rtt}_i}$ packets from flow i that traverse the router during the control interval. Therefore, each of these packets has to carry a negative congestion feedback of

$$n_i = \frac{\Delta R}{d} \cdot \text{H\_rtt}_i \cdot s_i \qquad (12.17)$$

In a control interval $h+\max\{-\phi, 0\}$ bytes must be deallocated by all XCP flows. Thus, the total decrease in the aggregated traffic rate is

$$\frac{h + \max\{-\phi, 0\}}{d} = \sum \frac{n_i}{\text{H\_rtt}_i} \qquad (12.18)$$

where the sum is considered over all packets in a control interval. Insert Equation (12.17) in Equation (12.18) it follows that (cf. Equation (12.11))

$$\Delta R = \frac{h + \max\{-\phi, o\}}{\sum s_i} = d \cdot \xi_n \tag{12.19}$$

Equations (12.12) and (12.13) maintain the self-clocking property of TCP for XCP. The difference of XCP to TCP is that an XCP sender is able to increase its congestion window by other values than 1 (or $1/\text{CWND}$) as a TCP sender does in slow start (or congestion avoidance) after the reception of an acknowledgment. And in contrast to TCP, an XCP sender is able to decrease its congestion window even after reception of an acknowledgment. Hence, an XCP sender can earlier and more accurately react on lower available bandwidth in a network than a TCP sender can do. In general, if XCP is used packet losses are rare events. But if packet loss events occur, a fallback procedure is used in which an XCP sender acts similar to a TCP sender.

12.3.3 Practicability of XCP

Implementing XCP in end systems is relative simple. Only slight changes in the source codes of TCP senders and TCP receivers must be done to make them XCP-capable. Equipping routers with XCP-capabilities is more costly but still simple. But the complexity of XCP in a router is comparatively high: For the most efficient implementation of the XCP algorithms (cf. appendix of [90, 91]) a couple of additions and 3 multiplications per packet are needed. Nevertheless, XCP is a promising candidate for improving congestion control in future IP-based networks.

XCP can be smoothly and incrementally deployed in current IP-based networks. Two cases have to be distinguished for that:

(1) some routers and receivers are not XCP-capable, and

(2) a mix of XCP and non-XCP connections coexist in the network.

In the first case, the XCP sender must check that all routers in the path and the receiver are XCP-capable. This can be done with existing TCP and IP mechanisms. If they are not XCP-capable, the XCP sender cannot use the XCP protocol and has to switch to a conventional transport protocol, e.g., TCP. In the second case, an XCP-capable router should be able to fairly handle both types of traffic, i.e., XCP flows should behave TCP-friendly. To reach this, an XCP-capable router has to distinguish between XCP and non-XCP traffic and queues them separately. Then the packets in both queues are processed such that XCP flows from the one queue reach the same average throughput than non-XCP flows from the other queue. This can be reached with a weighted fair queueing mechanism with dynamically adapted weights according to a TCP-Friendly Rate Control (TFRC) [95] approach (cf. [90, 91]).

12.4 Core-Stateless Fair Queueing (CSFQ)

The basic idea of Core-Stateless Fair Queueing [92] is to partition the network in islands of routers and to distinguish between the edge and the core of an island (cf. Figure 12.2). Each edge router estimates a

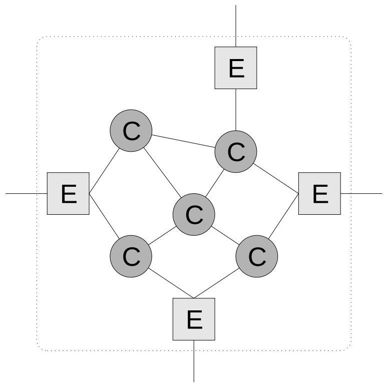


Figure 12.2: Island of edge (E) and core (C) routers with CSFQ-capabilities

per-flow rate for every incoming flow of packets that pass through the edge router into the CSFQ-capable network. For this calculation, the edge routers need to store per-flow states. The estimated per-flow rates are used to label the packets by inserting the estimated per-flow rate into each packet header of a flow. The core routers are equipped with FIFO queues and do not have to store any per-flow states. This property of the core routers is called core-stateless. The core routers perform a probabilistic dropping algorithm for arriving packets that is based on own measurements of the aggregated traffic and includes the information stored in the labels of the packets. The main goal of this dropping algorithm is to reach a fair share of bandwidth allocation between the flows passing a core router.

In the following Sections 12.4.1 and 12.4.2, the rate-estimation algorithm performed in an edge router and the packet-dropping algorithm performed in a core router are described. And in Section 12.4.3, some possible solutions how CSFQ-information can be carried in packets between edge and core routers are depicted. More details about CSFQ, e.g., pseudocodes for the algorithms used in edge and core routers or extensions to the here described basic CSFQ mechanism, can be found in [92].

12.4.1 CSFQ Estimation of Per-Flow Rates in an Edge Router

In an edge router the estimated flow rate $\widehat{r}_i$ of a flow i is updated every time a new packet of this flow arrives. This flow-rate estimation is done by using an estimation based on exponential averages. Let $t_i^{(k)}$ and $l_i^{(k)}$ be the arrival time and length of the k-th packet of flow i. Then $\widehat{r}_i$ is updated as follows:

$$\widehat{r}_i^{\text{new}} = (1 - e^{-T_i^{(k)}/K}) \cdot \frac{l_i^{(k)}}{T_i^{(k)}} + e^{-T_i^{(k)}/K} \cdot \widehat{r}_i^{\text{old}} \tag{12.20}$$

with $T_i^{(k)} = t_i^{(k)} - t_i^{(k-1)}$ and a constant value K. A rule of thumb for K (and the later used constants K_α and K_c) is that K should be set to a value that is approximately twice the maximum queueing delay. A packet of flow i is labeled with the latest update of $\widehat{r}_i$:

$$\text{label}_i = \widehat{r}_i \tag{12.21}$$

In addition to the per-flow rate estimation, the packet-dropping algorithm of CSFQ core routers described in the following Section 12.4.2 is also performed in an edge router.

12.4.2 CSFQ Packet-Dropping Algorithm in a Core Router

In a fluid-flow model the total aggregated arrival rate of n flows in a single link of a core router is

$$A = \sum_{i=1}^{n} r_i \tag{12.22}$$

Each of these flows should use a fair share rate α that the output link speed C of the core router is utilized:

$$C = \sum_{i=1}^{n} \min\{r_i, \alpha\} \tag{12.23}$$

Since the flows are packetized, A and α have to be estimated by $\widehat{A}$ and $\widehat{\alpha}$. Then the accepted aggregated traffic rate

$$F(\widehat{\alpha}) = \sum_{i=1}^{n} \min\{\widehat{r_i}, \widehat{\alpha}\} \tag{12.24}$$

of a core router can be also estimated.

The aggregated arrival rate of a core router is estimated based on exponential averages by

$$\widehat{A}^{\text{new}} = (1 - e^{-T/K_\alpha}) \cdot \frac{l}{T} + e^{-T/K_\alpha} \cdot \widehat{A}^{\text{old}} \tag{12.25}$$

with the inter-arrival time T between the current and the previous packet and a constant value K_α. An analogous formula is used for the estimated aggregated traffic rate $\widehat{F}$ accepted by a router.

If $\widehat{A} \geq C$ during the whole interval K_c, a link is assumed to be congested. If $\widehat{A} \leq C$ during the whole interval K_c, a link is assumed to be not congested.

A new value for the estimated fair share rate $\widehat{\alpha}$ is only computed after an interval in which the link has been classified as congested or not congested. If the link has been congested then $\widehat{\alpha}$ is updated as follows:

$$\widehat{\alpha}^{\text{new}} = \frac{C}{\widehat{F}} \cdot \widehat{\alpha}^{\text{old}} \tag{12.26}$$

If the link has been not congested then $\widehat{\alpha}^{\text{new}}$ is set to the largest rate of any active flow, i.e., to the largest label seen in a packet, during the interval K_c. In addition, two simple heuristics are used to limit the fluctuations between consecutive $\widehat{\alpha}$-calculations: (1) After every queue overflow, $\widehat{\alpha}$ is decreased by a small fixed fraction, e.g., 0.01, and (2) in order to avoid overcorrection, the decrease of $\widehat{\alpha}$ is limited by 0.25 of its previous value.

$\widehat{\alpha}$ is then used to calculate the probabilities for dropping packets of the flows. For flow i each incoming bit is dropped in a core router with the probability:

$$P_i = \max\left\{0, 1 - \frac{\widehat{\alpha}}{\widehat{r_i}}\right\} \tag{12.27}$$

If packets of flow i are dropped, the new rate r_i of flow i is changed to approximately $\widehat{\alpha}$, since the arrival-rate of flow i at the next core or edge router is approximately $\widehat{\alpha}$. Therefore, the packets of flow i should be re-labeled by

$$\text{label}_i^{new} = \min\{\widehat{\alpha}, \text{label}_i^{old}\} \tag{12.28}$$

Therefore, after a packet has past the edge router on the receiver-side of its flow the label of the packet contains the current flow rate the CSFQ-capable network part is able to provide. This information could be transferred to the receivers and sent back to the senders to finer tune the load the senders of the flows put into the network. But CSFQ does not provide such an explicit congestion-feedback mechanism. Only the packet losses in the edge and core routers of the CSFQ-capable network part are used to implicitly inform the senders about (impending) congestion in this part of the network. An extension of CSFQ with an additional explicit congestion-feedback mechanism is described in Section 12.5.

12.4.3 Carrying CSFQ-Information between Edge and Core Routers

The CSFQ-label can be carried inside the type-of-service (TOS) field in a normal IP header. With a 4m(antisse)4e(xponent) floating-point representation flow rates between 1 kbps and 65 Mbps can be stored in the TOS field with an accuracy of at least 6.25 % (cf. [92]). It is also possible to define an IP option in IPv4 or to introduce a hop-by-hop extension header in IPv6 to label packets according to the CSFQ mechanism.

12.5 Core-Stateless Fair Bandwidth Allocation for TCP (FBA-TCP)

FBA-TCP [93] uses the CSFQ mechanisms described in the last section to improve the congestion control of TCP connections. FBA-TCP works as follows: In edge routers of a network island (cf. Figure 12.3) FBA-TCP uses the same algorithms than CSFQ to estimate the rate of a flow and to label the packets of a flow with this estimated flow rate. In each edge and core router of a network island a fair share is

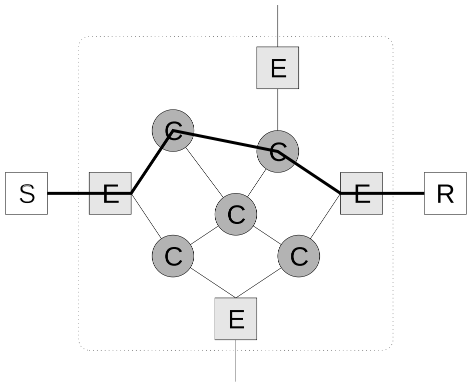


Figure 12.3: A single TCP connection traversing an island of routers with CSFQ-capabilities

estimated and packets of a flow are dropped according to Equation (12.27) if their label is larger than this fair share. This is exact the same algorithm used in CSFQ.

The new feature of FBA-TCP is that the edge router on the receiver-side of a flow do not remove the label from each packet. The edge router puts the label of a packet into a new option of an IPv4 header (or an extension header in IPv6) to transparently transfer this label to the TCP receiver through non-CSFQ-capable network parts. If the receiving end system of a TCP connection receives a packet with its label l or its estimated rate $\widehat{r}$, respectively, it delivers this value to the TCP receiver which computes a new allowed sending window for the TCP sender

$$\text{allowed sending window} = \text{RTT} \cdot \widehat{r} \tag{12.29}$$

using its own estimated RTT. Due to the full-duplex capability of TCP, i.e., a TCP receiver is simultaneously a TCP sender and vice versa, it is assumed that a TCP receiver possesses an adequate estimation of the RTT in the network path. The minimum of this sending window and the number of bytes the TCP receiver is currently able to accept from the TCP sender is the new advertised receiver window of the TCP receiver:

$$\text{advertised receiver window} = \min\{\text{allowed sending window}, \text{advertised receiver window}\} \tag{12.30}$$

This advertised receiver window is sent to the TCP sender inside the next TCP acknowledgment.

12.6 TCP Quick-Start (QS-TCP)

In contrast to all other approaches described in this chapter, TCP Quick-Start [94] uses feedback information from routers only at the beginning of a TCP connection to start this TCP connection with a larger initial congestion window if this is allowed by the routers in the network path. If the starting TCP connection is allowed by all routers in the network path to use a larger initial congestion window it performs a pacing mechanism to avoid sending its first segments in a burst.

The design principles of QS-TCP are:

- QS-TCP can only work if all routers in the path from the sending end system to the receiving end system are able to return explicit feedback.
- Only an underutilized router should allow a TCP sender to start with a larger initial congestion window than the standardized one. Otherwise, the queue at the router will show a transient behavior.
- Routers do not store any per-flow states to support QS-TCP. The only additional state a router has to store is its aggregated initial sending rate authorized over the most recent interval of time.

In the following Sections 12.6.1 and 12.6.2, the QS-TCP congestion-feedback transfer, its usage in the end systems, and the QS-TCP algorithm performed in the routers are described.

12.6.1 QS-TCP Congestion-Feedback Transfer and Usage in End Systems

If a TCP sender wants to use a larger initial congestion window it has to send a QS-TCP request (QSR) as a new IP option in its first IP packet which carries either the SYN or the SYN/ACK segment, respectively. The structure of a QSR is shown in Figure 12.4. The first two bytes of the QSR contain the IP option identifier and the length of this option (4 bytes). The third byte contains the QS-TCP time-to-live (TTL). It is randomly set by the TCP sender and decremented by one (modulo 256) from each router which approves the QSR. The QS TTL field is used by the sender to detect that all routers along the path

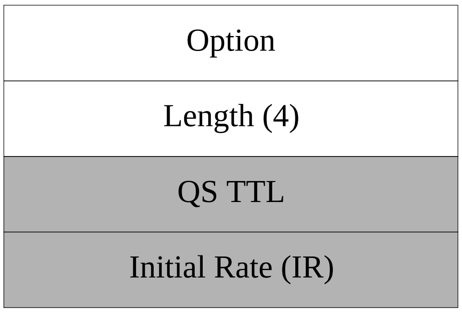


Figure 12.4: Quick-Start request option for IPv4

can cope with the QSR. In addition, the QS TTL byte is used to partly protect the network against a misbehaving TCP receiver which otherwise could forge the explicit feedback from the routers to enable the TCP sender to send with a larger initial sending window than the network routers can deal with. For this reason, the TCP sender calculates and stores the difference of the TTL in the IP header and the randomly chosen QS TTL:

$$\Delta\text{TTL} = \text{TTL} - \text{QS TTL} \pmod{256} \tag{12.31}$$

This value is later used by the TCP sender to validate the Quick-Start Response. The fourth byte of the QSR is the initial rate (IR) requested by the TCP sender. This byte is set by the TCP sender to its desired initial rate and decreased by QS-TCP-capable routers if necessary. The current proposal of QS-TCP (cf. [94]) sets the unit of this byte to packets per 0.1 seconds. Thus, the initial rate is limited to 2550 packets per second.

After receiving a QSR in a SYN or SYN/ACK segment, respectively, the TCP receiver returns a QS-TCP response (see Figure 12.5) to the sender. This response is carried as a new option in the TCP header in the SYN/ACK or ACK for the SYN/ACK, respectively. The first two bytes of this QS-TCP

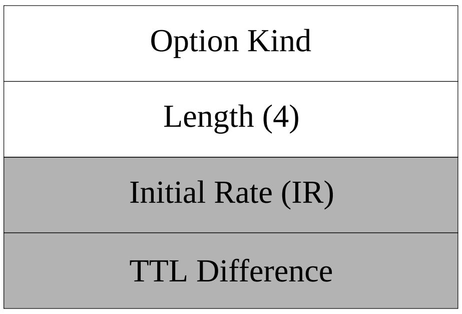


Figure 12.5: Quick-Start response option in the TCP header

response contain the TCP option identifier and the length of this option (4 bytes). The third byte of the QS-TCP response contains the initial rate allowed by the network. And the fourth byte of the QS-TCP response contains the difference of the TTL in the IP header and the QS TTL in the received IP packet with the carried SYN or SYN/ACK segment, respectively, calculated according to Equation (12.31).

After receiving a QS-TCP response, the TCP sender checks the validity of this response. The response is valid if it contains the correct value for the TTL difference and if the initial rate carried in the response is less or equal to the initial rate requested by the TCP sender. Due to this validity check, the negative influence of a misbehaving TCP receiver is limited to the initial rate requested by the TCP sender.

If the validity check for the QS-TCP response is successful, the TCP sender sets its initial congestion window to

$$\text{CWND} = \max\{\text{IR} \cdot \text{RTT} \cdot \text{MSS}, \text{default initial CWND}\} \tag{12.32}$$

using the initial rate from the response, the first measured RTT, and the maximum segment size (MSS). In order to avoid sending bursts of segments, the TCP sender has to perform a pacing mechanism. The QSR might not be able to use the fast path in routers, since routers have to handle these specific packets differently than normal IP packets. As a result, the first RTT measurement of a TCP sender might be larger than the real RTT between the sending and receiving end system. Therefore, the QS-TCP mode ends when the first acknowledgment arrives at the TCP sender. If in this point in time the congestion window has not been fully used, the congestion window is reduced to the current number of outstanding bytes before the first acknowledgment has been received.

If the validity check for the QS-TCP response is not successful, since the request has been failed or no QS-TCP response has been received at all, the TCP sender uses its default initial congestion window.

12.6.2 QS-TCP Algorithms in Routers

If a router receives an IP packet with a QSR, three different cases are possible:

- A router is not able to understand the QS-TCP request. In this case, the router forwards the QSR without changing any bytes in it.
- A router is able to understand the QS-TCP request but not willing or not in a position to accept a larger initial rate than the standardized one. In this case, the router either can delete the whole QSR from the IP packet or it can set the initial rate in the QSR to zero.
- A router is able to understand the QS-TCP request and is able to provide a larger initial rate than the standardized one. In this case, the router decrements the QS TTL by one, puts its own accepted initial rate in the IR byte of the QSR if its own initial rate is lower than the IR value stored in the QSR, and forwards the IP packet with the changed QSR included to the next hop.

How a router determines whether it can accept a larger initial sending rate than the standardized one or not depends on a newly introduced congestion control algorithm in the router that calculates the explicit router feedback. In [94], only some general rules for this congestion controller in the router are mentioned but no precise algorithm has been described. Therefore, the QS-TCP mechanism can be supported by routers in the network working internally with different congestion-control algorithms.

12.7 Summary

A summary and comparison of these new RCF approaches regarding their main properties and functionalities, requirements, applicability in the current Internet environment, complexity, expected performance gain, and possible combinations can be found in Chapter 14.

Chapter 13

Congestion Feedback from Routers: New Approaches

13.1 Introduction

In this chapter, two new RCF approaches, Fuzzy Explicit Window Adaptation (FEWA) and Enhanced TCP (ETCP), are described in detail.

13.2 Fuzzy Explicit Window Adaptation (FEWA)

In this section, a new EWA-related approach, called FEWA, is described whose queue-operating point is changed and whose calculation of the utilization factor α is based on a fuzzy controller.

13.2.1 FEWA Algorithm

The FEWA algorithm is similar to the EWA algorithm with two exceptions:

(1) The sending window (cf. Equation (12.1)) is calculated as follows:

$$\text{sending window} = \max\{\text{MSS}, \text{round}\,(\alpha \cdot \log_2(B - Q_i)) \cdot \text{MSS}\} \tag{13.1}$$

(2) And the utilization factor α (cf. Equation (12.2)) is calculated by a fuzzy controller only dependent on the current and the last but one measured queue length in a router:

$$\alpha = f(Q_i, Q_{i-1}) \tag{13.2}$$

Similar to EWA, the time of a control interval i of FEWA is set to 10 ms. But this value has to be adapted dependent on the bandwidths in the links the FEWA-capable router is connected with.

This fuzzy controller is related to the fuzzy controllers used in the Fuzzy Explicit Rate Marking Adaptation (FERMA) [96, 97] and in the more conservative FERMA-Modification (FERMAM) [98]. These approaches are located in Asynchronous Transfer Mode (ATM) switches and calculate congestion feedback which is used for the congestion control of Available Bit Rate (ABR) connections in ATM networks [4, 5, 6]. All linguistic rules and the membership functions of FERMAM are reused for FEWA and only a few parameters of the FERMAM fuzzy controller are adapted for FEWA.

13.2.1.1 General Fuzzy Controller

The most important part of a fuzzy control system is the fuzzy logic controller (FLC) [99, 100, 101]. With the FLC the human train of thoughts can be adapted in a simple way by using linguistic variables with their set of linguistic values and a small number of linguistic rules or relational expressions. One of the linguistic rules, linguistic rule R5, in the nonlinear FEWA fuzzy congestion controller (FCC) is for example (see Appendix E.1):

If the queue (length) is short and
the rate of change is increasing slowly, (R5)
then the utilization factor should be high.

"queue (length)", *"rate of change"*, and *"utilization factor"* are called linguistic variables; *"short"*, *"increasing slowly"*, and *"high"* are examples of their valid linguistic values. *"queue (length)"* and *"rate of change"* are the measured input variables of the linguistic rule while *"utilization factor"* is the output or control variable set by this linguistic rule.

The design of a FLC is split into two parts: First, the linguistic rules are set (surface structure), and second, the membership functions of the linguistic variables are determined (deep structure), quite often by a more intuitive and pragmatic choice.

Depending on the input parameter x the membership function m_V of a linguistic variable V denotes the weights $w_{v_k} \in [0,1]$ of the valid linguistic values v_k, $1 \leq k \leq n_V$:

$$m_V(x) = (w_{v_1}, \ldots, w_{v_{n_V}}) \in [0,1]^{n_V} \tag{13.3}$$

Here, each membership function m_V is represented by a composition of n_V membership functions of the fuzzy sets F_{v_k} of its corresponding linguistic values v_k:

$$\begin{aligned} m_V(x) &= m_{v_1}(x) \otimes \ldots \otimes m_{v_{n_V}}(x) \\ &= (w_{v_1}, \ldots, w_{v_{n_V}}) \end{aligned} \tag{13.4}$$

with the tensor operator $\otimes$. Let $\oplus$ denote the s-norm operator $\max$. For an FLC with a single output variable Y with the membership function $m_Y(y) = m_{y_1}(y) \otimes \ldots \otimes m_{y_{n_Y}}(y)$ its weighted membership function

$$\begin{aligned} m^*_Y(y) &= w_{y_1} \cdot m_{y_1}(y) \oplus \ldots \oplus \\ & \quad w_{y_{n_Y}} \cdot m_{y_{n_Y}}(y) \end{aligned} \tag{13.5}$$

represents the outcome of the L related fuzzy rules $v^{(l,1)} \times \ldots \times v^{(l,n)} \to y^{(l)}$, $1 \leq l \leq L$, with n linguistic values $v^{(l,1)}, \ldots, v^{(l,n)}$ of n linguistic variables $V^{(1)}, \ldots, V^{(n)}$ and their weights $w^{(l,1)}, \ldots, w^{(l,n)}$, dependent on the input-value vector $(x_1, \ldots, x_n)$ of the FLC, i.e., for $1 \leq l \leq L$, $1 \leq j \leq n$ and exactly one k, $1 \leq k \leq n_{V^{(j)}}$, it holds:

$$\begin{aligned} v^{(l,j)} &= v^{(j)}_k \\ w^{(l,j)} &= w_{v^{(j)}_k} = m_{v^{(j)}_k}(x_j) \end{aligned} \tag{13.6}$$

If none of the linguistic values $v^{(l,j)}$ of a linguistic variable $V^{(j)}$ are used in a linguistic rule l, i.e., the linguistic rule l is independent of $V^{(j)}$, then $w^{(l,j)}$ is set to 1.

Applying the often used norms $\min$ and $\max$ the weights w_{y_k}, $1 \leq k \leq n_Y$, of Y can be expressed as:

$$w_{y_k} = \max_{y^{(l)}=y_k} \{\min\{w^{(l,1)}, \ldots, w^{(l,n)}\}\} \tag{13.7}$$

The weights of all linguistic values of the linguistic variables are used to determine the required fuzzy value by a weighted analysis of the linguistic rules with a fuzzy inference engine. Due to computational simplicity the membership function of a linguistic variable is often triangular or trapezoidal shaped (cf. Figures 13.1, 13.2 and Appendix E.2).

13.2.1.2 FEWA Fuzzy Controller

At the beginning of every control interval i with a fixed duration, e.g., 10 ms, FEWA measures the current queue length Q_i and its current growth rate $G_i = Q_i - Q_{i-1}$ or its current fractional growth rate $\Delta G_i = G_i/B$, respectively. The membership function of FEWA regarding the queue length relative to the chosen target queue length QT is shown in Figure 13.1.

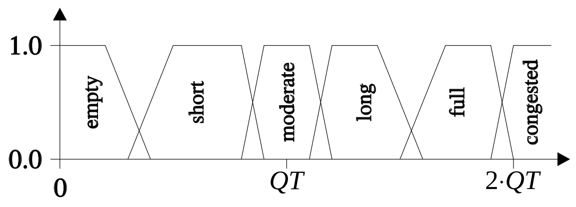


Figure 13.1: Membership function of the queue length Q of FEWA

The input range of the membership function of the fractional growth rate ΔG (see Figure 13.2) has been changed to substantial lower values because of the chosen target queue length $QT = \lfloor 0.25 \cdot B \rfloor$ at each router with FEWA-capabilities in the network.

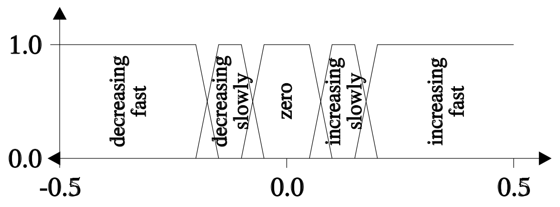


Figure 13.2: Membership function of the fractional queue growth rate ΔG of FEWA

Before every control interval i the weights of all linguistic values are set to 0. Then the weights for w_{Q_k}, $1 \leq k \leq 6$, for the six linguistic values ("empty", "short", "moderate", "long", "full", and "congested") of the current queue length Q_i and the weights w_{G_k}, $1 \leq k \leq 5$, for the five linguistic values ("decreasing fast", "decreasing slowly", "zero", "increasing slowly", "increasing fast") of the current fractional queue growth rate ΔG_i are determined. These weights are then used to calculate the weights w_{α_k}, $1 \leq k \leq 6$, for the linguistic values "very very little", "very little", "little", "medium", "high", and "very high" of the utilization factor α (cf. Appendix E.1). For the linguistic rule (R5), for instance, the last step of this computation can be expressed as (cf. Equation (13.7)):

$$w_{\alpha_5} = \max\{w_{\alpha_5}, \min\{w_{Q_2}, w_{\Delta G_4}\}\} \tag{13.8}$$

There may be other linguistic rules which have an influence on w_{α_5}. For FERMA, FERMAM, and for the fuzzy controller of FEWA, a different formula has been chosen to calculate the weights of the linguistic values of α. The linguistic rule (R5), for instance, can then be computationally expressed as:

$$w_{\alpha_5} = w_{\alpha_5} + w_{Q_2} \cdot w_{\Delta G_4} \tag{13.9}$$

There may be other linguistic rules which have an influence on w_{α_5}, too. By the weights w_{α_k}, $1 \leq k \leq 6$, the membership function m_α of α is converted to m^*_α (cf. Equation (13.5)). The weighted linguistic values of α are combined by a special fuzzy calculation, called Center-of-Gravity (COG), to the new utilization factor α (cf. [101]):

$$\alpha = \frac{\int y \cdot m^*_\alpha(y)\, dy}{\int m^*_\alpha(y)\, dy} \tag{13.10}$$

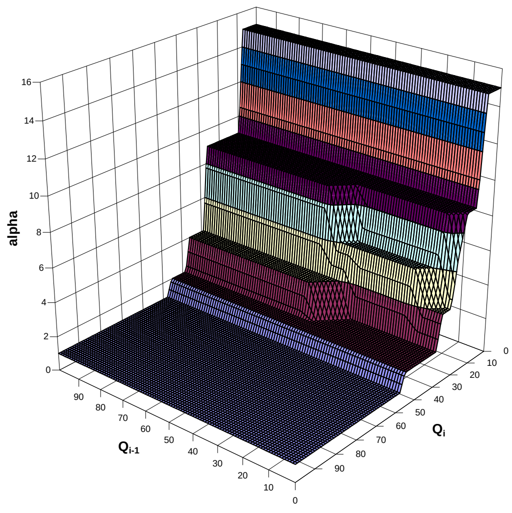
Analog to FERMA and FERMAM, the utilization factor α is determined by a simplified calculation where only the weights of the linguistic values of α are considered. Then, Equation (13.10) can be transformed into

$$\alpha = \frac{\sum\limits_{k=1}^{6} w_{\alpha_k} \cdot \alpha_k}{\sum\limits_{k=1}^{6} w_{\alpha_k}} \tag{13.11}$$

for a simplified computation with $\alpha_1 = 1.00$, $\alpha_2 = 2.00$, $\alpha_3 = 4.00$, $\alpha_4 = 6.00$, $\alpha_5 = 9.00$, and $\alpha_6 = 15.00$ in the current version of the FEWA fuzzy controller. The values for the α_k, $1 \leq k \leq 6$, are chosen to set the utilization factor α to a low value of 1 ($= \alpha_1$) if the queue is congested. If the queue is empty, the utilization factor is set to a high value of 15 ($= \alpha_6$). If the queue is neither congested nor empty, the utilization factor α is set to a value between 1 and 15 according to the linguistic rules of the FEWA FLC (cf. Appendix E.1).

Compared to the FLC of FERMAM, only these a_i's are adapted for the FLC of FEWA. Similar to FERMAM, these values are selected to achieve a moderate queue length Q in the neighborhood of the target queue length QT. But due to the more complex congestion control in IP-based networks compared to ATM networks and the less available information in IP routers compared to ATM switches, e.g., the number of TCP flows and their current congestion-control mode (slow start, congestion avoidance) are unknown, the target queue length in IP routers cannot be met as well as it can be done in ATM switches (cf. [97, 98]). Here, the precise values for the α's have been chosen both by general considerations and by evaluating various test simulations.

Figure 13.3 shows the current FEWA fuzzy controller's control surface for the utilization factor α and for the sending window with $B = 99$ packets and $QT = 24$ packets, respectively. The fuzzy controller of FEWA is developed in such a manner that the current queue length Q_i has the main influence on the calculation of the utilization factor α or of the sending window, respectively; the current (fractional) growth rate $(\Delta)G_i$ is used to refine this calculation (in Figure 13.3, Q_i has the dominant influence). For



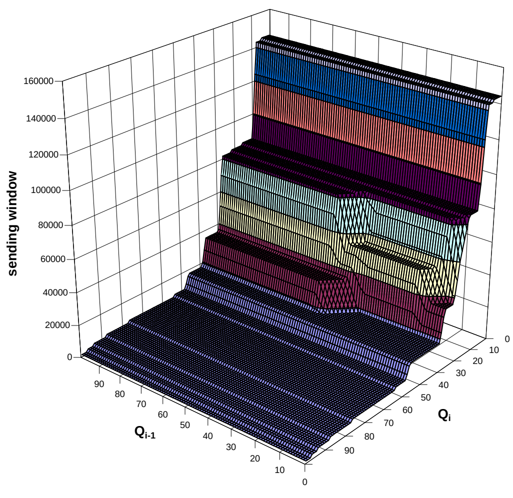


Figure 13.3: Control surfaces of the FEWA fuzzy controller for the utilization factor α and of the FEWA algorithm for the sending window

the FEWA algorithm the described membership functions of the linguistic variables (see Appendix E.2) and the values of the utilization factor are the result of a more intuitive and pragmatic choice and not of an analytic approach (that this can work is one of the main advantages of fuzzy-logic controllers compared to other controller types). Nevertheless, this selection provides promising results.

It should be noted that due to the differences in the congestion-feedback functions of FERMAM and FEWA, in the FEWA fuzzy controller the values for the parameters α_i cannot be independently chosen from the maximum queue length B in a router. Therefore, these fuzzy controller parameters have to be carefully determined for each maximum queue length B in FEWA-capable routers in a network. For other maximum queue lengths B than 99, the α_i's should be selected that the new control surface for the sending window patterns the control surface for the sending window shown in Figure 13.3. A rule of thumb for this α_i-selection can be found in Appendix E.3.

13.2.2 Practicability of FEWA

In a real IP router there is no need for a complex and computational intensive fuzzy inference engine in the FLC. After the linguistic rules have been found and the linguistic values are tuned by a simulator, the control surface is known and can be stored as a lookup table for selected sampling points requiring only a few kilobytes of read-only memory (ROM) in a FEWA-capable router. In combination with a simple interpolation algorithm FEWA can be implemented in such a way with a very fast response time.

13.3 Enhanced TCP (ETCP)

EWA and the related FEWA approach are developed to improve TCP's end-to-end congestion control by using TCP's built-in flow control mechanism. The same mechanism is used by Core-Stateless Fair Bandwidth Allocation for TCP (FBA-TCP, cf. Section 12.5). If a standard TCP sender is assisted by

(F)EWA or FBA-TCP it limits its sending window to the minimum of its current congestion window and its current advertised receiver window. This advertised receiver window has been possibly decreased by a (F)EWA- or FBA-TCP-capable router if necessary due to (impending) congestion in the router. Therefore, a standard TCP sender in combination with one of these approaches can faster react on (impending) congestion in this router. But feedback information about additional available bandwidth in such a router can often not be used by the sender to increase its sending window as fast as it decreases its sending window in the case of (impending) congestion, particularly, if the TCP sender is in the slow start phase with a small congestion window or in the congestion avoidance phase.

If all routers—or at least the bottleneck routers—in a network are equipped with (F)EWA- or FBA-TCP-capabilities the TCP senders receive accurate and useful information about the current network load. Then, the overall performance of (F)EWA and FBA-TCP can be significantly increased if the congestion-control algorithm in a standard TCP sender is slightly changed by adapting the semantic of the advertised receiver window stored in every TCP acknowledgment: The current advertised receiver window of a TCP sender is not just an upper bound for its sending window or congestion window (cf. [37]), respectively, it is used to calculate the new congestion window more appropriately. This new TCP congestion-control mechanism based on a finer evaluation of the advertised receiver window stored in TCP acknowledgments is called Enhanced TCP (ETCP).

After the reception of an acknowledgment, the TCP congestion-control mechanisms slow start and congestion avoidance are not used any longer in ETCP to increase the congestion window. This is the task of the new ETCP congestion-control mechanisms interpreting the current advertised receiver window. But after a segment loss, basic TCP congestion-control mechanisms (cf. Section 2.3.2.6.1) as well as fast retransmit / fast recovery (cf. Section 2.3.2.6.2) are used as fallback procedures also in ETCP. Thus, for a single ETCP connection its history is a less attached value to the congestion control than it is for a single TCP connection.

In the following, three ETCP variants are considered which differ in the calculation of the new congestion window:

- **ETCP 1:**
  The new congestion window of an ETCP sender is set to the current advertised receiver window:
  $$\text{CWND}(T_k) = \text{ARWND}(T_k) \tag{13.12}$$
- **ETCP 2:**
  A TCP sender performing slow start doubles its congestion window after it has received CWND acknowledgments. This property of TCP is imitated by ETCP. Thus, the new congestion window of an ETCP sender converges to its doubled value after the ETCP sender has received CWND acknowledgments:
  $$\text{CWND}(T_k) = \min\left\{\text{CWND}(T_{k-1}) \cdot 2^{1/\text{CWND}(T_{k-1})}, \text{ARWND}(T_k)\right\} \tag{13.13}$$
- **ETCP 3:**
  Making an ETCP sender more aggressive—if allowed by the network—than in its variant 2, another strategy can be used. Here, the new congestion window of an ETCP sender converges to the

current advertised receiver window after the ETCP sender has received CWND acknowledgments:

$$\text{CWND}(T_k) = \begin{cases} \text{ARWND}(T_k) & \text{if } \text{CWND}(T_{k-1}) \geq \\ & \text{ARWND}(T_k) \\ \text{CWND}(T_{k-1}) \cdot & \\ (\text{ARWND}(T_k)/\text{CWND}(T_{k-1}))^{1/\text{CWND}(T_{k-1})} & \text{else} \end{cases} \tag{13.14}$$

A variation of this ETCP variant is the following:

$$\text{CWND}(T_k) = \begin{cases} \text{ARWND}(T_k) & \text{if } \text{CWND}(T_{k-1}) \geq \\ & \text{ARWND}(T_k) \\ \text{CWND}(T_{k-1}) \cdot & \\ (\text{ARWND}(T_k)/\text{CWND}(T_{k-1}))^{1/\text{IW}} & \text{if } \text{ARWND}(T_k) > \\ & \text{IW} > \text{CWND}(T_{k-1}) \\ \text{CWND}(T_{k-1}) \cdot & \\ (\text{ARWND}(T_k)/\text{CWND}(T_{k-1}))^{1/\text{CWND}(T_{k-1})} & \text{else} \end{cases} \tag{13.15}$$

This restricts the aggressiveness of an ETCP sender to its initial aggressiveness in cases of small current congestion windows. As a result, the ETCP congestion control is more robust if packet losses in the network can occur that are not caused by congestion in (RCF-capable) routers. But such a variation of the ETCP variant 3 is not considered in this dissertation.

After the first acknowledgment (the SYN/ACK) of a connection has been received by the ETCP sender, the ETCP sender knows the first ARWND that reflects the current load conditions in the network path (cf. remarks on TCP window scale option [30] in Section 2.3.2.5). This first ARWND is used in (1) to set the first valid CWND to this value. The ETCP variants 2 and 3 use this first ARWND together with the initial CWND to calculate the first valid CWND according to the explained formulas. The following Figure 13.4 shows the schematic CWND process of the different ETCP variants compared to standard TCP for a constant example-ARWND of 64 segments and dependent on the number of consecutively received acknowledgments with the SYN/ACK included.

In all these ETCP variants, a pacing mechanism according to the pacing algorithm of FS-TCBI (cf. Section 5.2.5.2) is performed. This is mainly used to avoid a bursty sending behavior of an ETCP sender in cases where the CWND has been largely increased. In addition, such a pacing mechanism makes the congestion control of an ETCP sender more resistant to different round trip times.

In the following Figure 13.5, the CWND process of a single ETCP connection is compared with the CWND process of a single standard TCP connection using an example-network path with a round trip time of 100 ms traversing a single router. In this specific network scenario the differently controlled ETCP connections reach throughputs of 132.42, 109.76, and 125.20 segments per second while the standard TCP connection reaches a throughput of 112.25 segments per second. The ETCP variants 1 and 3 are able to outperform standard TCP. But the ETCP variant 2 reaches a slightly lower throughput compared to standard TCP. The reason for this decreased performance of ETCP variant 2—at least in this

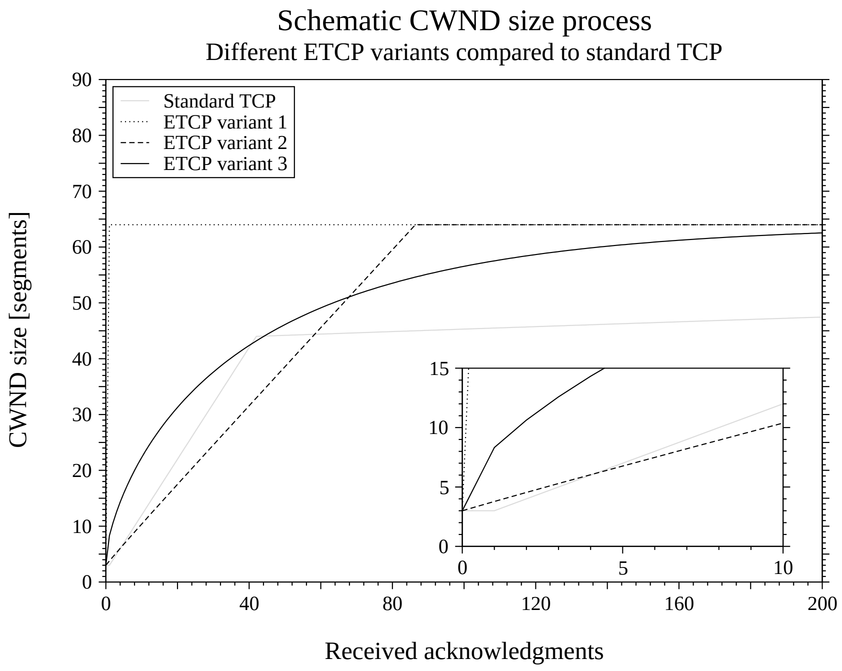


Figure 13.4: Schematic CWND process of the different ETCP variants compared to standard TCP

network scenario—is its too conservative CWND-calculation particularly at the beginning of a connection. It can be expected that the ETCP variant 2 outperforms standard TCP if longer connections are considered.

A very important difference between the first ETCP and the other two ETCP variants is that the ETCP variants 2 and 3 keep the self-clocking property of standard TCP, only some modifications are

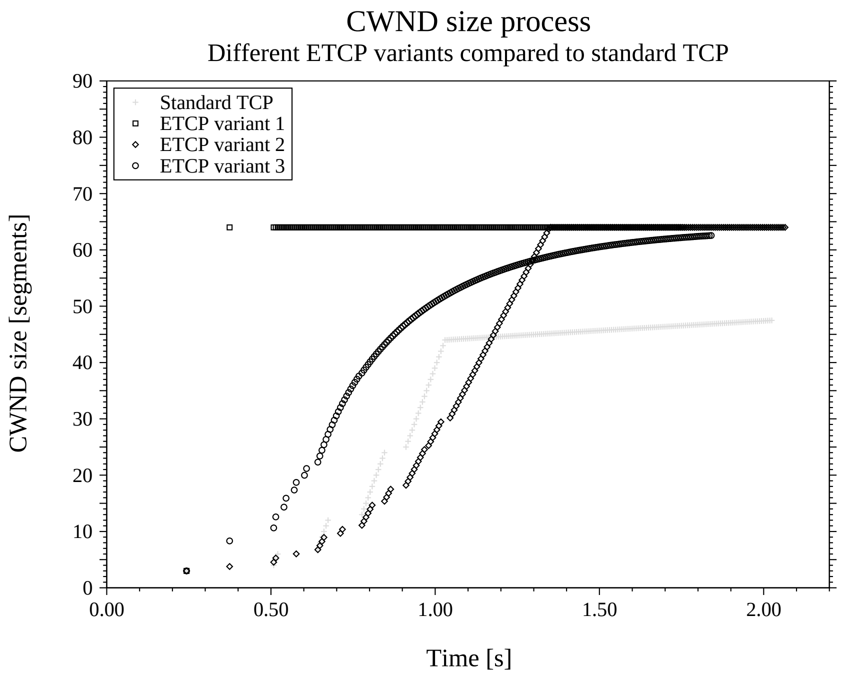


Figure 13.5: CWND process of the different ETCP variants compared to standard TCP

done how the CWND is increased by the ETCP sender after an acknowledgment has been received. As a result, even in combination with the pacing mechanism the ETCP variant 1 might be too aggressive to the network in load scenarios with concurrent traffic. This has been evaluated by test simulations. In addition, the ETCP variant 3 reaches the best performance in these test simulations. Therefore, the results of the performance investigations of ETCP shown in Chapter 17 are obtained by using the ETCP variant 3.

If the bottleneck router is not equipped with (F)EWA or FBA-TCP capabilities, the ETCP senders do not receive adequate information about the current load in the network. In this worst-case scenario, an ETCP sender following the first ETCP variant is allowed to send too many segments that congestion in the network is likely to occur. The other ETCP variants, and particularly the second one, are more robust in this case.

Since EWA and FEWA are developed to work with standard TCP, the use of ETCP might have an influence on the load observed in (F)EWA-capable routers. Thus, if ETCP is used in end systems either the (F)EWA control algorithms, the pacing algorithm in the end systems, or the congestion-window calculations used in the different ETCP variants have to be revised to avoid too largely loaded or even congested routers in the network.

Currently, ETCP is only a promising idea how congestion control can be improved in future IP-based networks by using slightly adapted existing TCP congestion- or flow-control mechanisms. Further research has to be done to investigate how ETCP influences the traffic characteristics in the network and how a distributed congestion management can be advantageously used together with ETCP. In addition, due to the changed semantic of the advertised receiver window in ETCP, the original end-to-end flow control of TCP is confined or even eliminated. Which effects this can have in existing or future TCP-based applications has to be carefully investigated.

13.4 Summary

A summary and comparison of these new RCF approaches regarding their main properties and functionalities, requirements, applicability in the current Internet environment, complexity, expected performance gain, and possible combinations can be found in the following Chapter 14.

Chapter 14

Congestion Feedback from Routers: Comparison and Applicability of Existing and New Approaches

14.1 Introduction

In the last two chapters, different RCF approaches have been described in detail. This chapter compares these RCF approaches under several aspects. The remainder of this chapter is organized as follows. Section 14.2 compares the RCF approaches with regard to their main properties and functionalities, requirements, and expected performance gain. And in Section 14.3, the applicability of these RCF approaches in the current Internet environment is considered in detail. Finally, Section 14.4 concludes this chapter by selecting some of these RCF approaches for further performance investigations in the following chapters and by substantiating this choice.

14.2 Comparison

14.2.1 Introduction

This section contains a summary and comparison of the main properties and functionalities, requirements, and expected performance gain of the RCF approaches described in the last two chapters.

14.2.2 Properties, Functionalities, Requirements, and Expected Performance Gain

The following Table 14.1 shows the main properties and functionalities, requirements, and expected performance gain of the considered RCF approaches of the second type. Which of these approaches should be used in future IP-based networks depends on the desired level of compatibility with the currently deployed transport protocols TCP and UDP in the end systems. XCP, for example, seems to be the most powerful approach to improve the overall performance of a network—at least if the network is a high-speed network. But XCP requires slight but necessary adaptations of the transport protocol in the end

Table 14.1: Summary and comparison of the main properties and functionalities of different RCF mechanisms in IP-based networks

	(F)EWA	ETCP	XCP	CSFQ	FBA-TCP	QS-TCP
supports TCP / UDP flows	yes / no	yes / no	no / no	yes / yes	yes / no	yes / no
is transparent for senders / receivers	yes / yes	no / yes	no / no	yes / yes	yes / no	no / no
direction of the RCF approach	one-way	one-way	two-way	one-way	one-way	two-way
provides explicit / implicit RCF	yes / yes	yes / yes	yes / yes	no / yes	yes / yes	yes / yes
RCF used to increase / decrease sending rate	no / yes	yes / yes	yes / yes	no / yes	no / yes	yes / yes
provides continuous RCF	yes	yes	yes	yes	yes	no
needs per-flow state in some routers	no	no	no	yes	yes	no
needs symmetrical routing	(yes)[1]	(yes)[1]	no	no	no	no
uses new IP / TCP header (option)	no / no	no / no	yes / no	no / no	yes / no	yes / yes
has built-in pacing mechanism	no	yes	no	no	no	yes
applicable in the current Internet[2]	yes	(yes)	no	yes[3]	(yes[3])	no
add. complexity in end systems / routers	low / mid	low / mid	low / high	low / high	low / high	low / mid
expected performance gain[4]	(+)++	++++	+++++[5]	++	+++	+

systems.

[1]Section 12.2.4 contains some remarks about a mechanism that does not require symmetrical routing

[2]Here it is considered if a RCF approach is transparent for end systems (or only minor changes in end systems are necessary) and can be stepwise deployed, i.e., the RCF approach does not require new functionalities in all routers of the Internet, for further details see Section 14.3 and Table 14.3

[3]The expected performance gain of these approaches is limited if only a few routers or end systems in the Internet are CSFQ- or FBA-TCP-capable

[4]Compared to standard TCP

[5]At least in high-speed networks (the performance of XCP in other network scenarios, e.g., in low- to medium-speed (radio) access networks, is currently unknown)

In addition, the overhead per packet in the XCP-capable routers seems to be feasible but large compared to other approaches.

In contrast to XCP, (F)EWA does not require any changes in the end systems. But (F)EWA is less powerful than XCP, since with (F)EWA the sending window of TCP senders cannot be as accurately controlled as with XCP. ETCP as an extension of (F)EWA is able to compensate this in parts at the cost of slightly adapted TCP senders. The basic (F)EWA and ETCP approaches require a symmetrical routing at least in the bottleneck router(s) of the network. But if this necessary condition cannot be guaranteed in the network, (F)EWA and ETCP can be easily adapted to work with a new TCP option or an IP option header (cf. Section 12.2.4). CSFQ does not provide an explicit feedback for (TCP) senders. It is mainly developed to increase the fairness between flows in a network (part). Thus, its performance gain is rather limited. But FBA-TCP as an extension of CSFQ might be a potential candidate for an improved congestion control in IP-based networks. From its design FBA-TCP should have a comparable power to increase the efficiency in a network than (F)EWA. The main disadvantage of FBA-TCP is that the edge routers in the CSFQ-capable part of the network have to store per-flow information to label the packets of a flow. As a result, this approach can only work if the number of flows passing an edge router is rather low. In a sense, the main difference of XCP and FBA-TCP is that XCP labels packets in the transport protocol of a sending end system while FBA-TCP (re)labels packets in the routers of a CSFQ-capable network part.

QS-TCP is able to allow TCP connections to start with a larger initial congestion window than standardized if possible. This aim of QS-TCP is too limited to deploy QS-TCP as the (main) approach to improve congestion control in future IP-based networks. The most interesting aspect of QS-TCP is the mechanism how the network can be saved from misbehaving TCP receivers that forge the congestion-control feedback from routers in order to enable their senders to start sending with a too large and not allowed initial congestion window.

Table 14.2 shows which of the RCF approaches considered in this dissertation can be combined with each other.

Table 14.2: Possible combinations of RCF approaches

can be combined with:	(F)EWA	ETCP	XCP	CSFQ	FBA-TCP	QS-TCP
(F)EWA		yes	in parts[5]	yes	yes	yes
ETCP	yes		no	yes	yes	yes
XCP	in parts[5]	no		yes	no	no
CSFQ	yes	yes	yes		yes	yes
FBA-TCP	yes	yes	no	mandatory[6]		yes
QS-TCP	yes	yes	no	yes	yes	

[5]It is planned to replace the Effi ciency Controller of XCP with a FEWA-related fuzzy controller

[6]FBA-TCP is based on CSFQ

14.2.3 Summary

A-priori, the most promising candidate of existing router congestion feedback mechanisms to improve congestion control in IP-based networks is XCP. Therefore, the XCP performance should be investigated in more detail for different network scenarios and variable traffic loads.

It has been analytically shown that the algorithms of the fairness controller of XCP are well chosen. But the efficiency controller of XCP could be suboptimal. To investigate other algorithms for this part of XCP, e.g., a modified fuzzy controller of the FEWA mechanism or a CSFQ-related estimation of the aggregated rate in edge or core routers, might be promising. In addition, the overall performance of XCP or its adaptations could be improved if a pacing mechanism in an XCP sender is introduced. Since the XCP approach has a much higher complexity per packet in the routers than ETCP, for example, it might be interesting to investigate also the performance of the ETCP approach compared to standard TCP and XCP or its adaptations, respectively.

XCP and ETCP do not support UDP flows. If also UDP flows should be optionally supported by a router congestion feedback mechanism in future IP-based networks, these mechanisms have to be extended to operate with rates as input information for the routers and changes in rates as congestion-feedback information from the routers instead of windows. This approach can be combined with a CSFQ-related packet loss criterion to limit the rate of flows that are not able to understand or to penalize flows that are not willing to accept this congestion feedback from the network. Both the flow information for the routers and the feedback information from the routers can be carried in a new IP option if IPv4 is used or in an IP extension header if IPv6 is used, respectively. The protocol instances running in the end systems are then responsible to set and evaluate the values in this IP option.

14.3 Applicability in the Current Internet Environment

14.3.1 Introduction

The focus of this section is on the applicability of these RCF approaches in the current Internet environment. The section describes if and how these RCF approaches can be (gradually) deployed in the current Internet in order to improve the performance of the network. In addition, it is explained if and how these RCF approaches can deal with the security mechanisms of Internet Protocol Security (IPSec) [29, 12, 102].

In this section, each of the different RCF approaches is considered in more detail and with regard to the following point of views:

- Some of the considered RCF approaches require several network properties and changes in routers and/or end systems. Which of these requirements can be easily provided and which of these requirements are hard to reach in the current Internet environment? This viewpoint includes also the consideration of the additional complexity a RCF approach requires in the routers and/or in the end systems.

- Is it possible at all to gradually deploy the considered RCF approach in the current Internet environment? If this question can be approved there are other points which should be considered then:

  - ○ How large is the influence on the expected performance gain of the RCF approach if only a few routers in the Internet, e.g., all routers in a network part, are equipped with additional RCF functionalities?
  - ○ If the expected performance gain of a RCF approach is negatively influenced by a gradual deployment, how can this negative influence on the expected performance gain be limited to reach an improved expected performance even with such a successive deployment? Here, it is considered which strategy should be used to choose the routers in the Internet that are primarily equipped with the new functionalities of the considered RCF approach.
  - ○ In addition, it is worth to investigate which RCF approaches are suited for deployment in parts of the Internet, e.g., a provider subnetwork, to improve congestion control at least in these parts of the Internet.

- Another interesting aspect is if and how a RCF approach is able to deal with existing security mechanisms in the Internet, e.g., IPSec. In IPSec, two different encryption methods can be distinguished:

  - ○ In the transport mode of IPSec, the payload of an IP packet is encrypted. This mode provides a secure transfer of transport-layer data between two end systems. In this case, an intermediate router is not able to read or write fields in the TCP header.
  - ○ In the tunnel mode of IPSec, a complete IP packet is encrypted and carried as payload of a new IP packet. This mode provides a secure transfer of IP packets between two routers, for example, to establish a virtual private network (VPN) in the Internet. In this case, an intermediate router is not able to find out how many separate IP flows are carried inside such an IP tunnel. Also in this case, an intermediate router is not able to read or write fields in the header of TCP segments transferred in some of these IP flows.

In the following Sections 14.3.2 to 14.3.7 each of the RCF approaches is considered with respect to the above stated viewpoints. Finally, Section 14.3.8 summarizes these considerations.

14.3.2 (Fuzzy) Explicit Window Adaptation ((F)EWA)

EWA and its improvement FEWA can only work if TCP segments and their TCP acknowledgments pass the same (F)EWA-capable routers in the network. Therefore, both RCF approaches can be only deployed in the current Internet if IP packets are symmetrically routed in the part of the network where routers are equipped with (F)EWA. Since a (F)EWA-capable router must be able to decrease the advertised receiver window in TCP acknowledgments to inform TCP senders about the current load in the router, parts of the TCP header must be changeable. Thus, (F)EWA in its basic variant cannot be deployed in the current Internet if (some of the) TCP connections are encrypted using the IPSec transport mode.

Both stated necessary conditions of supporting (F)EWA in the current Internet can be eliminated if the basic (F)EWA mechanism of manipulating TCP's built-in flow control is replaced by a mechanism based on a new option in an IP header or based on a new IPv6 extension header, respectively, if IPv6 will be widely deployed in future. Then, the congestion feedback information of the routers can be transferred in the IP header option or IPv6 extension header of an IP packet carrying a TCP segment. In addition, this (F)EWA variant can cooperate with the IPSec transport mode. But (F)EWA cannot be used together with the IPSec tunnel mode. This can be only reached if it is ensured by new IPSec mechanisms that the IP header option or the IPv6 extension header for (F)EWA is not encrypted in the IP tunnel and carried with the encapsulated IP packet after the IP tunnel. Since this (F)EWA variant requires that the receivers are able to handle the new IP option or new IPv6 extension header, the transparency of (F)EWA for receiving end systems is lost.

The main advantage of (F)EWA compared to other RCF approaches is that (F)EWA can be gradually deployed in the network. If this successive deployment is accurately done, i.e., the bottleneck routers in the network are primarily equipped with (F)EWA, the expected performance gain of (F)EWA is not reduced. In addition, (F)EWA can be separately supported in a network part, e.g., a provider subnetwork, to improve the congestion control and the overall performance of TCP connections at least in this network part. In addition, the complexity of the (F)EWA algorithm in the routers is comparatively low compared to XCP, for example.

14.3.3 Enhanced TCP (ETCP)

ETCP depends on the implementation of the FEWA algorithms in all routers or at least in the bottleneck routers of the network. Therefore, all statements about the applicability of FEWA in current Internet routers and the transport of FEWA congestion control information from routers to the end systems are valid for ETCP. A disadvantage of ETCP in its current version compared to, for example, QS-TCP is that an ETCP-capable sender is not able to verify whether the routers in a network are equipped with FEWA-capabilities or not. Thus, it is not possible to deploy ETCP senders in a network where not all bottleneck routers are equipped with FEWA. But with mechanisms similar to those used in QS-TCP, it is possible to provide ETCP senders with the feature to detect if routers are equipped with FEWA capabilities and agree with the new advertised receiver window carried in the TCP acknowledgments or not. Then, ETCP can be understood as an extension of QS-TCP where congestion feedback from routers is carried to a TCP sender during the whole lifetime of its connection. In addition, the shortcoming of the QS-TCP proposal [94] that no router algorithms are specified in detail is lapsed with this ETCP adaptation.

Since ETCP requires slight but fundamental changes (e.g., a pacing mechanism) in the congestion-control algorithm of each TCP sender, this RCF approach can only be hardly completely deployed in the current Internet. But it is possible to gradually update existing TCP senders with this new improved congestion-control functionality after all (bottleneck) routers in the network have been equipped with FEWA.

So far, an open question is how the changed semantic of the advertised receiver window in ETCP senders influences the traffic characteristics in the Internet regarding, for example, the queue dynamics in the routers or the load in the access networks. This has to be carefully evaluated by simulations (cf. Chapter 17).

14.3.4 Explicit Control Protocol (XCP)

Although XCP has been identified as the most promising RCF approach in the last chapter, the deployment of XCP in the current Internet is hard to reach, since end systems and routers have to be equipped with new congestion-control algorithms. Particularly the complexity of these new control algorithms in routers seems to be high compared to other RCF approaches. An XCP-capable router, for example, has to perform several additions and multiplications per packet.

In theory, XCP can be smoothly and incrementally deployed in current IP-based networks. Two cases have to be distinguished for that: (1) some routers and receivers are not XCP-capable and (2) a mix of XCP and non-XCP connections cooperate in the network. In the first case, the XCP sender must check that all routers in the path and the receiver are XCP-capable. This can be done with existing TCP and IP mechanisms. If at least one of these routers is not XCP-capable, the XCP sender cannot use the XCP protocol and has to switch to a conventional transport protocol, e.g., TCP. In the second case, an XCP-capable router should be able to fairly handle both types of traffic, i.e., XCP flows should behave TCP-friendly. To reach this, an XCP-capable router has to distinguish between XCP and non-XCP traffic and queues them separately. Then the packets in both queues are processed such that XCP flows from the one queue reach the same average throughput than non-XCP flows from the other queue. This can be reached with a weighted fair queueing mechanism with dynamically adapted weights according to a TCP-Friendly Rate Control (TFRC) [95] approach (cf. [90, 91]). Thus, in practice the gradual deployment of XCP in the current Internet environment is still a challenging problem.

Note that the quality of a gradual deployment of XCP in the current Internet is different from the quality of a gradual deployment of (F)EWA. For example, if at least one router in the network path is not XCP-capable, i.e., XCP cannot be used, the whole benefit of XCP compared to standard TCP gets lost. On the other hand, if a few routers in a network path are not (F)EWA-capable the performance gain of (F)EWA compared to standard TCP can be maintained if at least the bottleneck routers in the network path are equipped with (F)EWA. Thus, a noticeable performance gain of XCP in a network can only be expected if most of the routers and end systems are deployed with the XCP algorithms.

XCP uses an additional congestion header for carrying congestion-control information from XCP senders to XCP-capable routers and congestion feedback information in the inverse direction. If this congestion header can be separated from the remaining data of the transport protocol and excluded from the encryption of the transport protocol data (IPSec's transport mode), XCP is not affected at all by transport-layer security mechanisms in the current Internet. But XCP cannot be used together with the tunnel mode of IPSec (cf. notes on IPSec tunnel mode in Section 14.3.2).

14.3.5 Core-Stateless Fair Queueing (CSFQ)

CSFQ is mainly developed to improve the fairness between flows in a network (part) of CSFQ-capable routers. Therefore, it is possible to gradually deploy CSFQ in the current Internet. But CSFQ does not provide an explicit congestion feedback from routers to further improve the end-to-end performance of flows. Thus, its usability for flows which only traverse small CSFQ-capable network parts is somehow limited.

The main disadvantage of CSFQ compared to all other considered RCF approaches is that the edge routers of a CSFQ-capable network (part) need to store per-flow states in order to identify flows, to measure their current flow rate, and to mark packets of a flow according to the current measured flow rate. Thus, CSFQ is limited in its scalability and its applicability is restricted to network parts with a small number of flows traversing an edge router.

The identification of single flows in an edge router can be complicated or even prevented if some of these flows entering a CSFQ-capable network (part) are encrypted. If, for example, transport-layer flows between two end systems are encrypted using the transport mode of IPSec, the edge router is not able to separate these flows from each other. Then, the edge router can only determine the rate of the combined transport-layer flows between two end systems. And if flows of IP packets between two routers outside the CSFQ-network are encrypted in the tunnel mode of IPSec, the edge router can only determine a rate for this combined flow of IP flows. Thus, there may exist different types of flows inside a CSFQ-capable network. How CSFQ should work in this case is not considered by the developers of CSFQ. In the end, the CSFQ approach cannot be used with flows encrypted by IPSec.

14.3.6 Core-Stateless Fair Bandwidth Allocation for TCP (FBA-TCP)

FBA-TCP is an extension of CSFQ that provides an explicit feedback from a CSFQ-capable network to improve congestion control of TCP senders. Therefore, all properties and boundaries of CSFQ are valid for FBA-TCP. In addition, since the explicit feedback from the CSFQ-capable network has to be transferred to the TCP receiver, the FBA-TCP approach cannot cooperate with IPSec's tunnel mode.

14.3.7 TCP Quick-Start (QS-TCP)

As already mentioned, QS-TCP does not specify any algorithms in the routers, only some general design principles and rules for these algorithms are given (cf. Section 8 in [103], [94]). But it can be assumed that the implementation complexity of QS-TCP in routers is comparable to that of (F)EWA. QS-TCP is not transparent for end systems. It requires minor changes in TCP receivers and more substantial changes in TCP senders. Therefore, the deployment of QS-TCP in the current Internet is harder to reach than the deployment of (F)EWA, for example. And this comes with a limited expected performance gain of QS-TCP compared to (F)EWA, for example, since QS-TCP is only used to start a TCP connection with a larger initial congestion window than it is prescribed by the standard. In addition, a gradual deployment of QS-TCP in (parts of the) network has comparable limitations in usefulness and expected performance gain than a gradual deployment of XCP (cf. Section 14.3.4).

QS-TCP is able to cooperate with Internet security mechanisms based on IPSec's transport mode. But QS-TCP is not able to cooperate with encapsulated IP flows as it is used in the tunnel mode of IPSec.

14.3.8 Summary

In the previous sections, it has been described if and how the different RCF approaches (F)EWA, ETCP, XCP, CSFQ, FBA-TCP, and QS-TCP can be deployed in the current Internet environment. The focus of this consideration has been on the additional complexity of the RCF approaches in end systems and routers, their capability to be gradually deployed in the current Internet, their expected performance loss if they are only gradually deployed, and their ability to cooperate with security mechanisms in the Internet like IPSec. Table 14.3 summarizes these considerations.

Table 14.3: Applicability of different RCF mechanisms in the current Internet environment

	(F)EWA	ETCP	XCP	CSFQ	FBA-TCP	QS-TCP
add. complexity in end systems / routers	low / mid	low / mid	low / high	low / high	low / high	low / mid
can be gradually deployed	yes	yes	yes	yes	yes	yes
performance losses if gradually deployed[1]	(no[2])	yes	yes	(no[3])	(no[3])	yes
can cooperate with IPSec transport / tunnel mode	(yes[4]) / no	(yes[4]) / no	(yes[5]) / no	no / no	no / no	yes / no

Although XCP is the most promising approach in improving the performance of the network, its capability to be (gradually) deployed in the current Internet is limited. Therefore, implementing XCP in the Internet is rather a medium- to long-term task than a near-term one. The next best expected RCF approach regarding the performance gain is ETCP. But since ETCP requires some changes in TCP senders, its complete deployment in the current Internet is also hard to reach. In addition, the influence of ETCP senders on the current traffic characteristics in the Internet has been not investigated so far. If both the ability of a gradual deployment and the expected performance gain of a RCF approach is taken into account, FEWA in its variant with a new IP option or an IPv6 extension header should be considered as the first choice for a near-term improvement of the performance in the Internet. FBA-TCP has a comparable expected performance gain than FEWA, but the shortcomings of FBA-TCP are that per-flow state is required in some routers and that FBA-TCP cannot be used together with IPSec mechanisms in the current Internet environment. CSFQ and QS-TCP have limited expected performance gains compared to the other RCF approaches. In addition, CSFQ cannot be used in cooperation with IPSec mechanisms and a gradual deployment of QS-TCP in the current Internet environment is hard to reach. Therefore, CSFQ and QS-TCP should not be considered as the primary choice of a RCF approach in the Internet. But it is conceivable to use a QS-TCP-related approach working over the whole lifetime of a TCP connection in combination with, for example, FEWA to benefit from properties of both approaches. Then, the slightly adapted QS-TCP sender is able (1) to faster increase its sending window to the currently available bandwidth and (2) to accordingly decrease its sending window to the current

[1]Compared to a full deployment of a RCF approach in the Internet

[2]If at least the bottleneck router is equipped with (F)EWA

[3]If the bottleneck router is in the CSFQ-capable part of the network

[4]Only if the variant with a new IP header option or a new IPv6 extension header is used

[5]If the XCP congestion header is not part of the encrypted transport data

load in the network path than it is possible with a standard TCP sender.

14.4 Conclusion and Outlook

In this section, it has been considered if and how different RCF approaches can be (gradually) deployed in the current Internet environment. In addition, it is investigated if and how these RCF approaches can cooperate with existing security mechanisms like IPSec.

Since there exist a trade-off between the capability to be simply (and gradually) deployed in the current Internet environment and the expected performance gain a RCF approach has, it is decided to select the further and in more detail considered RCF approach with regard to the following two aspects: (a) easy (gradual) deployment in the current Internet environment and (b) maximum expected performance gain. Therefore, the following RCF mechanisms are chosen for a detailed investigation of their performance:

(a) **FEWA (ETCP, extended QS-TCP+FEWA)**
FEWA can be simply deployed in the current Internet environment, since the end systems can be kept unchanged and the additional algorithms in the FEWA-capable routers have a low complexity. In addition, it is sufficient to equip only the bottleneck routers in a network (part) with FEWA capabilities, i.e., FEWA can be gradually deployed to reduce or even prevent congestion in a network (part).

The expected performance gain of FEWA in networks with a high dynamical load is moderate compared to standard TCP. Feedback information about a suddenly decreased available bandwidth in a FEWA-capable router can be transferred to the TCP senders within a half round trip time. The TCP senders are then able to faster and more appropriate react on (impending) congestion in the network. Feedback information about an increased available bandwidth in such a router can also be transferred within a half round trip time. But this information is only used to perform standard TCP's probing mechanisms for available bandwidth (slow start or congestion avoidance) in the sending end systems. Therefore, FEWA is mainly used to reduce or prevent congestion in the network and not to higher utilize the links in the network (although this is done if packet losses due to congestion are significantly reduced). But if all routers or at least the bottleneck routers in a network are equipped with FEWA, this shortcoming of FEWA can be eliminated if the ARWND-semantic of standard TCP in the sending end systems is replaced by the new ARWND-semantic of ETCP. Alternatively, FEWA can be combined with ideas related to QS-TCP to extend TCP's network-probing mechanisms with an explicit non-congestion feedback from the routers that allows such a slightly adapted TCP sender to be more aggressive than a standard TCP sender if it is allowed by all routers in the network path.

(b) **XCP (FXCP)**
The main disadvantage of XCP is that it has the highest complexity compared to the other RCF approaches. In addition, XCP cannot be gradually deployed in parts of a network. If at least one router or end system in a network path is not able to cope with XCP, XCP is not performed and standard TCP is used instead of it.

But XCP promises the highest expected performance gain compared to the other RCF approaches—at least in high-speed networks. Its performance in other network scenarios like (radio) access networks with much lower available bandwidth has to be carefully investigated. Nevertheless, XCP should be investigated as an a-priori best case for the performance gain that can be reached with existing RCF approaches in IP-based networks. In addition to this investigation, in future it will be intended to evaluate the performance of an XCP adaptation where the efficiency controller of XCP in the routers is replaced by a fuzzy-based controller related to the controller used in FEWA. This new XCP-variant is called FXCP.

In this dissertation, from these selected RCF approaches FEWA, ETCP together with FEWA-capable routers, and XCP are investigated in a network with wired links. Their performance results are compared with standard TCP and TCP assisted by EWA. The other chosen RCF approaches, e.g., FXCP, are investigated in fixed, wireless, and mobile networks in an ongoing industrial project in cooperation with Ericsson Research.

Chapter 15

Congestion Feedback from Routers: Performance-Evaluation Overview

15.1 Introduction

This chapter describes and explains the calculations of the performance metrics used for the performance evaluation of standard TCP compared to the RCF approaches EWA, FEWA, ETCP, and XCP between Internet routers and end systems. In addition, the simulation model used for the performance evaluation of standard TCP and these RCF approaches is depicted in detail.

15.2 Performance-Evaluation Metrics

In this section, the different metrics used for the performance evaluation of several RCF approaches are described.

15.2.1 Throughput

The two mean throughput metrics defined in Section 2.3.2.7 are used to compare the overall throughput performance of RCF approaches with standard TCP. These two throughput metrics are calculated separately for each (W)LAN by considering all TCP, ETCP, or XCP connections with receivers in the receiving end systems located in each of the (W)LANs.

15.2.2 Number of Packet Losses

During the simulated time also the number of packet losses in the base stations of the (W)LANs is counted. These values can then be used to calculate the mean number of packet losses or the mean packet loss rates, respectively, for the base stations of the (W)LANs as additional performance metrics.

15.3 Simulation Model

In this section, the basic simulation model including the considered network topology with the location of (F)EWA- or XCP-capable routers and the traffic model are described. This simulation model is used in the next two chapters to evaluate the performance of the different RCF approaches EWA, FEWA, ETCP, and XCP compared to standard TCP. Chapter 16 considers the performance of the RCF approaches EWA and FEWA which are transparent for TCP instances in end systems. And Chapter 17 is related to the performance investigation of RCF approaches which require minor changes in the TCP senders in the sending end systems or even completely replace TCP.

15.3.1 Network Topology

The hierarchically structured network topology of the simulation model is shown in Figure 15.1. It consists of a public Internet part, one core network (CN), two radio access networks (RANs), and six radio cells (RCs) with a (wireless (W)) LAN access technology.

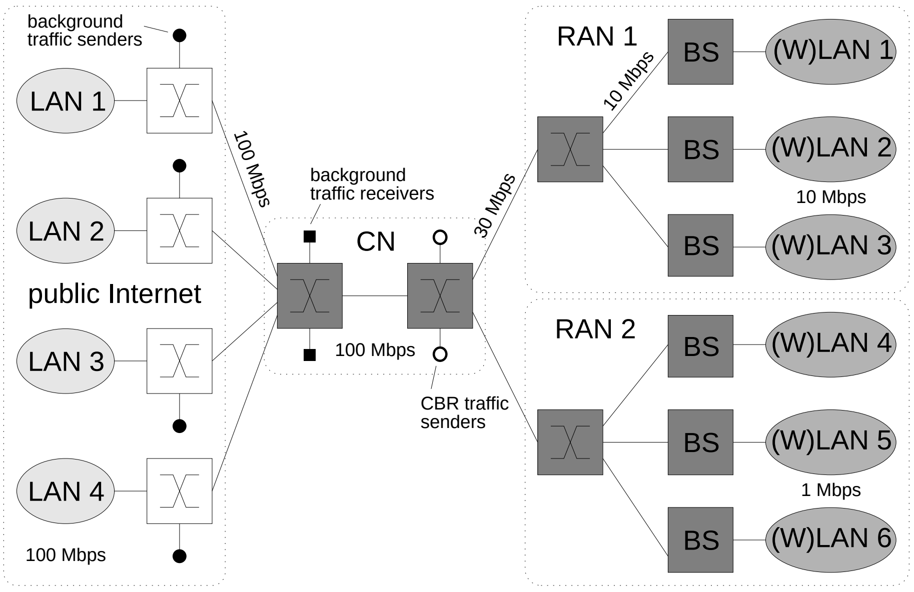


Figure 15.1: Structure of the simulated network topology

The public Internet part of the simulation model consists of four LANs and four routers. Each LAN has a bandwidth of 100 Mbps and a delay of $5 \cdot 10^{-7}$ seconds. All links in the public Internet have a bandwidth of 100 Mbps. The delay between the routers in the public Internet and the ingress router of the core network is set to a fixed value of 50 ms to adjust a mean round trip time for the simulations. The link between the ingress router and the egress router of the core network has a bandwidth of 100 Mbps and a delay of 5 ms. The ingress routers of the RANs are connected to the egress router of the core network with links with a bandwidth of 30 Mbps and a delay of 5 ms.

The radio access networks RAN 1 and RAN 2 are intended to represent wireless access networks, e.g., WLAN. Such a setup is very interesting, since this represents one of the most important scenarios where a large-bandwidth core network is connected with a small-bandwidth radio access network, resulting in bottlenecks at the transition between these networks. Therefore, the RAN routers connected to the (W)LANs 1–6 represent “base stations” ((W)LAN routers) of these networks. The base stations are connected to the ingress RAN router by links with a bandwidth of 10 Mbps and a delay of 5 ms. Each (W)LAN in RAN 1 provides a bandwidth of 10 Mbps while each (W)LAN in RAN 2 provides a bandwidth of 1 Mbps. These values are chosen to reflect typical maximum bandwidths of currently widely used WLAN access technologies. The delays in the (W)LANs are set to a constant, small value. Both the packet error rate p in the (W)LANs and the handover-rate of the mobile receiving end systems in which the TCP or XCP receivers are located is adjustable. With this simulation model, the performance of different RCF approaches, e.g., standard TCP, standard TCP assisted by EWA (TCP+EWA), standard TCP assisted by FEWA (TCP+FEWA), ETCP, or XCP can be investigated in different scenarios that reflect the properties of various future IP-based networks. But in this dissertation, the performance of standard TCP, standard TCP either assisted by EWA or assisted by FEWA, ETCP, and XCP is investigated in a network scenario with reliable links in the (W)LANs, immobile receiving end systems, and only the mean network load is varied.

All router queues used in the simulations are droptail queues. These router queues have a maximum queue length limit of 100, i.e., every router queue can store at most $B = 99$ IP packets. Thus, the target queue length of FEWA is set to $QT = 24$ packets.

In each of the LANs in the public Internet, n TCP, ETCP, or XCP senders are located. Their TCP or XCP receivers are distributed over receiving end systems located in the (W)LANs of the two radio access networks. 16 background TCP senders are connected to the routers in the public Internet, 4 to each of them. The background TCP receivers are located in receiving end systems which are connected to the ingress router of the core network. 12 Background UDP senders are located in sending end systems which are connected to the egress router of the core network, while their UDP receivers are uniformly distributed over receiving end systems in the (W)LANs of the RANs.

15.3.2 Location of NIS-Capable Routers

Both routers in the core network and the routers in the radio access networks are equipped with several NIS mechanisms based on RCF. These additional router functionalities can be optionally switched on and off that the performance of standard TCP, TCP+EWA, TCP+FEWA, ETCP combined with FEWA, and XCP can be investigated in the simulation model.

15.3.3 Traffi c-Load Models

Properly characterizing traffic loads for interactive Internet users is a difficult undertaking. In order to represent some mixture of different applications, it has been decided to use one TCP-, ETCP-, or XCP-based application class to generate WWW traffic and one UDP-based application class to generate voice traffic.

The first application class uses TCP, ETCP, or XCP connections where the number of segments to send is determined by a WWW traffic model [81]. This traffic model is derived from real HTTP (version

1.0) traces in corporate and educational environments and uses three abstraction levels: The session level, the page level, and the packet level. Here, a simplified version of this model is used which consists only of the first two levels. In every WWW session a log-normally distributed number of WWW pages is sent with page sizes that follow a curtailed Pareto distribution, i.e., the page size is limited to 1 MByte. The time between the pages, i.e., the inter-connection or reading time, is gamma distributed. The load in the network can be easily adjusted by using the exponentially distributed session inter-arrival time with a different parameter. The distributions and parameters chosen for the stochastic variables of the simplified WWW traffic model are shown in Table 7.1.

Table 15.1: Distributions and parameters for the stochastic variables of the simplified WWW model

Stochastic variable	Distribution	Distribution parameter(s)
Inter-session time	Exponential	$\mu = 5.0$ s
Pages per session (pps)	Lognormal	$\mu =$ 25.807 pps $\sigma = 78.752$ pps
Inter-page time (reading time)	Gamma	$\mu =$ 35.286 s $\sigma = 147.390$ s
Page size	Pareto (limited)	$\alpha = 1.7584$ $\beta = 30458$ Bytes

Each WWW page is transported by a single TCP connection from a sending end system (server) to a receiving end system (client). The TCP connections used to transfer consecutive pages of a session do not overlap in time, since the reading or inter-page time is started after the previous page has been completely transferred.

This traffic model is used for all TCP, ETCP, or XCP connections whose receivers are located in receiving end systems in the (W)LANs of the RANs. The background traffic TCP connections in the public Internet part of the simulation model use the same distribution for their pages per session and their page size as shown in Table 7.1. But the inter-session time and the inter-page time are deterministically set to zero.

The second application class includes UDP data streams which send every 20 ms a UDP datagram with a payload length of 36 bytes. This class is used to model traffic with a constant bit rate (CBR) like voice-over-IP.

15.3.4 Implementation Details

The whole simulation model is implemented in ns-2 (version 2.1b9) [61]. Since the TCP header in ns-2 does not contain an advertised receiver window, it has been included. For all TCP connections in the simulation model, a modified ns-2 implementation of a TCP NewReno [11] sender or receiver is used. The modifications in the TCP senders include the analysis of the advertised receiver window of received TCP acknowledgments, the optional use of the ETCP algorithms, and additional statistical performance-evaluation methods (cf. Section 15.2). In addition, the limitation of the CWND on the current ARWND (cf. Equation (2.8)) is also implemented in the TCP senders used in the simulation model. The TCP receivers in ns-2 are modified to set the advertised receiver window in TCP acknowledgments according to their current capability to receive new TCP segments (then the advertised receiver window is set to a large constant value of 373760 bytes (= 256 TCP segments with a maximum segment size of 1460 bytes) to not influence the different RCF algorithms working in the routers) or to perform the Freeze-TCP [104] mechanism before a handover is executed (then the advertised receiver window is temporarily set to zero to stop transmissions from the TCP sender during the handover procedure in order to avoid packet losses due to the handover). In the latter case, also the whole mobility model is implemented in the TCP receivers, since Freeze-TCP needs an estimation of the exact point in time where the next handover will be executed to inform the TCP sender early enough about the impending handover.

For XCP connections in the simulation model, the original XCP ns-2 implementation of the XCP algorithms working in routers and end system is used. This implementation has to be adapted to include it in the simulation model. In addition, some errors in the original ns-2 XCP implementation regarding the header size have been corrected.

What is not considered in the simulation model is the amount of extra-bytes necessary for some RCF approaches, e.g., the size of the congestion header of XCP transferred in every XCP segment and acknowledgment or the three bytes of the TCP window scale option transferred in each TCP segment and acknowledgment. The reason to ignore these extra-bytes is that the size of XCP's congestion header is not defined and that the TCP window scale option might not be used, since it is optional. This simplification—if it is a simplification at all—can be done, since the consideration of a couple of extra-bytes in each segment or acknowledgment would increase the per-link delay of a segment or an acknowledgment by only a few microseconds. The influence of several such small extra-delays on the overall round trip time of a connection is negligible (in the simulation model the overall extra-delay would cause an approximately 0.05 % higher overall round trip time).

In contrast to a real mobile node there is no possibility to detect an impending handover within the network simulation, e.g., by monitoring the signal strength. Because of this restriction the point in time of the next handover event, the handover time T_{ho}, is determined by the TCP receiver itself. Within the time interval $T_{\text{ho}} - T_{\text{now}} \leq \text{SRTT}$ the receiver has the possibility to freeze the TCP sender where T_{now} is the current simulation time and SRTT is the smoothed round trip time measured by the TCP receiver. In the current implementation of Freeze-TCP, it is necessary that the TCP receiver receives TCP segments that it can acknowledge. Otherwise, the TCP sender cannot be "frozen". An additional mechanism that a TCP receiver is able to retransmit the last sent TCP acknowledgment with an adapted advertised receiver window of zero to immediately freeze the TCP sender is not introduced so far.

In the basic Freeze-TCP mechanism described in [104], three copies of the last acknowledgment prior to the disconnection are sent to inform the TCP sender about the re-established connection. Unfortunately, the authors of Freeze-TCP give no information what to do in case of loosing the wakeup acknowledgments. This has been identified as a possible problem. Therefore, the wakeup-mechanism of Freeze-TCP has been extended by including a timeout timer to retransmit these three acknowledgments if the TCP receiver does not receive any TCP segments until this timer timeouts. The number of retransmissions and the timeout interval are adjustable by the user. The default timeout interval is set to twice the SRTT which seems to be an appropriate compromise between the avoidance of unnecessary retransmissions and a reaction in short time due to packet losses. For the simulations, it is tried to wakeup the

TCP sender after the completion of the handover for at most 64 times.

In order to count packet losses in queues and links, the link queue class and the error model class of ns-2 are adapted. This adapted link queue class is also able trace the queue length and to generate histograms of the queue length.

Unfortunately, the queues in ns-2 belong to the links and they are all output queues from a router's point of view. Therefore, it has been necessary to create a new basic router class that virtually combines all queues which belong to a specific router in the network topology. Since a single queue of a link in ns-2 can belong to different objects of the new router class, a specific handling of packets is needed in each of the queues. To reach this, each incoming IP packet is marked in a queue that the router class is able to distinguish between IP packets coming from and going to different neighbored network nodes. For this reason, the link-trace class in ns-2 is adapted.

The new basic router class provides three key functionalities: (a) a mechanism to monitor the packet queue and to calculate statistical values, e.g., the smoothed queue length, (b) an algorithm to calculate a router-internal advertised receiver window for each link, and (c) a mechanism that adapts the advertised receiver window in the TCP acknowledgments. As a derivation of the basic router class, the EWA router class is implemented in ns-2. This EWA router class performs the EWA algorithm described in [89] for all of the combined queues in the router class. With one exception: It has been detected by evaluating results from earlier simulations that it is advantageous to introduce an upper bound for the parameter α or for the sending window of the EWA-algorithm. Otherwise, routers which are low-loaded for a longer period of time will increase their sending window without a limit In the simulations, an upper bound for the sending window is used that is equal to the maximum advertised sending window (373760 bytes) of the TCP receivers. In reality, the parameter α should be limited (cf. Section 12.2.4).

New RCF approaches related to EWA (cf. Section 12.2.4) can be easily implemented in ns-2 by deriving the basic router class. Examples are FEWA and FEWA-based approaches [105] specifically designed for radio network controllers (RNCs) in WCDMA-based networks like UMTS [24]. But also other RCF approaches, e.g., XCP can be supported by the router class if minor changes in the basic mechanisms of this router class are performed. Therefore, the existing simulation model can be easily extended to evaluate also the performance of other and in future developed RCF approaches in IP-based networks.

For the traffic model of the simulation, a new traffic generator class is implemented. This traffic generator class provides WWW traffic based on TCP or XCP, ON-OFF traffic based on TCP or XCP, CBR traffic based on UDP, and UDP-based traffic according to trace files, e.g., real video or audio traffic traces.

Chapter 16

Performance Evaluation of EWA and FEWA by Simulations

16.1 Introduction

In the following, two simulation scenarios are considered which differ in their mean load in the network: the first simulation scenario uses $n = 24$ TCP senders in each LAN of the public Internet to generate WWW traffic in the network, the second one doubles this value. In each scenario the performance metrics of standard TCP are compared with the performance metrics of standard TCP assisted by EWA or standard TCP assisted by FEWA. In addition, for each simulation scenario the queue length process, the queue length histogram, and the complementary cumulative distribution function (CCDF) of the queue length of each downstream queue in the base stations in RAN 2 are shown.

Currently, twenty simulation runs per scenario case are performed. Every run for each simulation scenario with either a lower or a higher mean network load considers a simulated time of 18000 seconds. The transient analysis of the simulations (cf. Appendix C) shows that the throughputs of the TCP connections have a transient phase of about 1800 seconds. Therefore, only the throughput observations after the transient phase can be used to calculate the mean values of the different throughput metrics. In addition, also the mean numbers of packet losses in the downstream queues of the base stations in RAN 2 are calculated by ignoring packet losses during the transient phase.

The last column Δ in the following Tables 16.1 and 16.2 denotes whether the simulation results of one approach compared to the simulation results of another approach are statistically significantly different or not. In this column the approaches are compared in the following order: standard TCP $\leftrightarrow$ TCP+EWA, standard TCP $\leftrightarrow$ TCP+FEWA, and TCP+EWA $\leftrightarrow$ TCP+FEWA. A '-' denotes that the first approach of the comparison reaches significantly better results while a '+' denotes that the second approach reaches significantly better results. And a '=' means that with the simulation results no significant difference between the two compared approaches can be concluded. The details of the statistical evaluation of these simulation results can be found in Appendix D.

16.2 Scenario 1: TCP NewReno, TCP+EWA, and TCP+FEWA, Lower Mean Network Load

Table 16.1: Simulation results of scenario 1

		standard TCP	TCP+EWA	TCP+FEWA	Δ
$\overline{TPO}$ of TCP connections [segments/second]	(W)LAN 1	52.77	52.87	52.83	$===$
	(W)LAN 2	52.78	52.90	52.69	$===$
	(W)LAN 3	53.04	52.84	52.89	$===$
	(W)LAN 4	37.78	37.86	38.68	$=++$
	(W)LAN 5	37.72	37.79	38.78	$=++$
	(W)LAN 6	37.82	38.04	38.65	$=++$
$\overline{TPC}$ of TCP connections [segments/second]	(W)LAN 1	47.59	47.66	47.67	$===$
	(W)LAN 2	47.65	47.67	47.63	$===$
	(W)LAN 3	47.71	47.67	47.66	$===$
	(W)LAN 4	36.36	36.29	36.46	$==+$
	(W)LAN 5	36.36	36.31	36.46	$==+$
	(W)LAN 6	36.36	36.42	36.39	$===$
mean number of packet losses in base station ((W)LAN router)	(W)LAN 1	0.00	0.00	0.00	$===$
	(W)LAN 2	0.00	0.00	0.00	$===$
	(W)LAN 3	0.00	0.00	0.00	$===$
	(W)LAN 4	2581.10	954.30	0.00	$+++$
	(W)LAN 5	2598.30	912.70	0.00	$+++$
	(W)LAN 6	2561.50	891.55	0.00	$+++$

In this scenario with a lower mean network load the base stations in RAN 1 are not congested at all, i.e., no packets are lost in the downstream queues of these base stations, and the base stations in RAN 2 are rarely congested if standard TCP connections are considered. The throughputs of TCP connections traversing the base stations in RAN 1 are comparable for the approaches standard TCP, TCP+EWA, and TCP+FEWA; EWA reaches nearly the same throughput than standard TCP while the throughput loss of FEWA is approximately 0.12 % compared to standard TCP and EWA. For connections traversing the base stations in RAN 2, TCP+FEWA outperforms standard TCP as well as TCP+EWA in most of the throughput metrics. For these TCP connections, the overall mean throughput gain of TCP+FEWA is approximately 2 % compared to both standard TCP and TCP+EWA.

Compared to standard TCP, the EWA algorithm considerably decreases the number of packet losses in congested routers by more than 64 %. But packet losses can be completely prevented in these routers if the EWA algorithm is replaced by the FEWA algorithm.

16.3 Scenario 2: TCP NewReno, TCP+EWA, and TCP+FEWA, Higher Mean Network Load

Table 16.2: Simulation results of scenario 2

		standard TCP	TCP+EWA	TCP+FEWA	Δ
$\overline{TPO}$ of TCP connections [segments/second]	(W)LAN 1	47.72	47.79	47.94	$===$
	(W)LAN 2	47.72	47.67	47.91	$===$
	(W)LAN 3	47.97	47.67	47.94	$-==$
	(W)LAN 4	28.31	28.04	29.30	$=++$
	(W)LAN 5	28.10	27.96	29.48	$=++$
	(W)LAN 6	28.17	28.11	29.42	$=++$
$\overline{TPC}$ of TCP connections [segments/second]	(W)LAN 1	44.98	44.97	45.09	$===$
	(W)LAN 2	44.94	44.96	45.03	$===$
	(W)LAN 3	45.06	44.92	45.04	$===$
	(W)LAN 4	30.46	30.22	30.34	$===$
	(W)LAN 5	30.42	30.14	30.45	$-==$
	(W)LAN 6	30.35	30.22	30.40	$===$
mean number of packet losses in base station ((W)LAN router)	(W)LAN 1	0.00	0.00	0.00	$===$
	(W)LAN 2	0.00	0.00	0.00	$===$
	(W)LAN 3	0.00	0.00	0.00	$===$
	(W)LAN 4	14840.70	4636.05	0.00	$+++$
	(W)LAN 5	15196.55	4706.90	0.00	$+++$
	(W)LAN 6	14980.55	4626.10	0.00	$+++$

In this scenario with a higher mean network load the base stations in RAN 1 are still not congested at all, i.e., no packets are lost in their downstream queues, and the base stations in RAN 2 are much more often congested than in the previous scenario if standard TCP connections are considered. For standard TCP connections traversing the uncongested base stations in RAN 1 the throughputs are comparable to the throughputs of TCP connections either assisted by EWA or assisted by FEWA (TCP+FEWA reaches a slight throughput gain compared to the other two approaches). For TCP connections traversing the base stations in RAN 2 TCP+FEWA outperforms both other approaches. For these TCP connections the overall mean throughput gain of TCP+FEWA is approximately 4 % compared to standard TCP and approximately 5 % compared to TCP+EWA. The connection-oriented mean throughput is slightly increased by both approaches TCP and TCP+FEWA compared to TCP+EWA.

Compared to standard TCP, the EWA algorithm considerably decreases the number of packet losses in congested routers by 69 %. But also in this scenario packet losses can be prevented if the EWA algorithm is replaced by the FEWA algorithm in these routers.

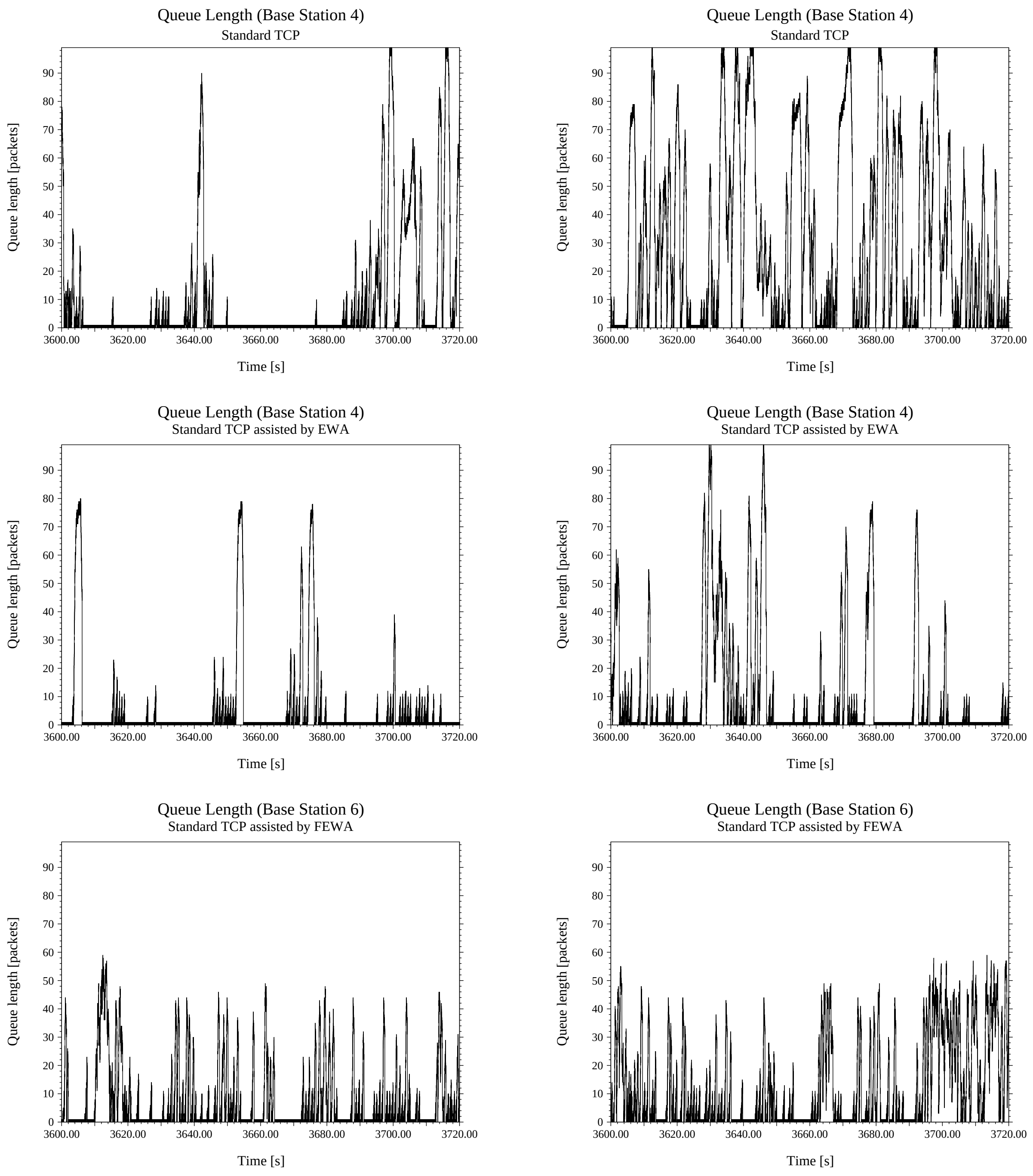


Figure 16.1: Queue length process in the downstream queue of a base station in RAN 2 for standard TCP, TCP+EWA, and TCP+FEWA (lower mean network load on the left side, higher mean network load on the right side)

16.4 Queue Lengths in Bottleneck Base Stations

In this section, the queues of the base stations in RAN 2 are considered in more detail. Figure 16.1 shows typical queue length processes in the downstream queue of a base station in RAN 2 for both simulation scenarios and for an observation interval of 120 s.

In the scenario with a lower mean network load and compared to standard TCP, TCP+EWA is able

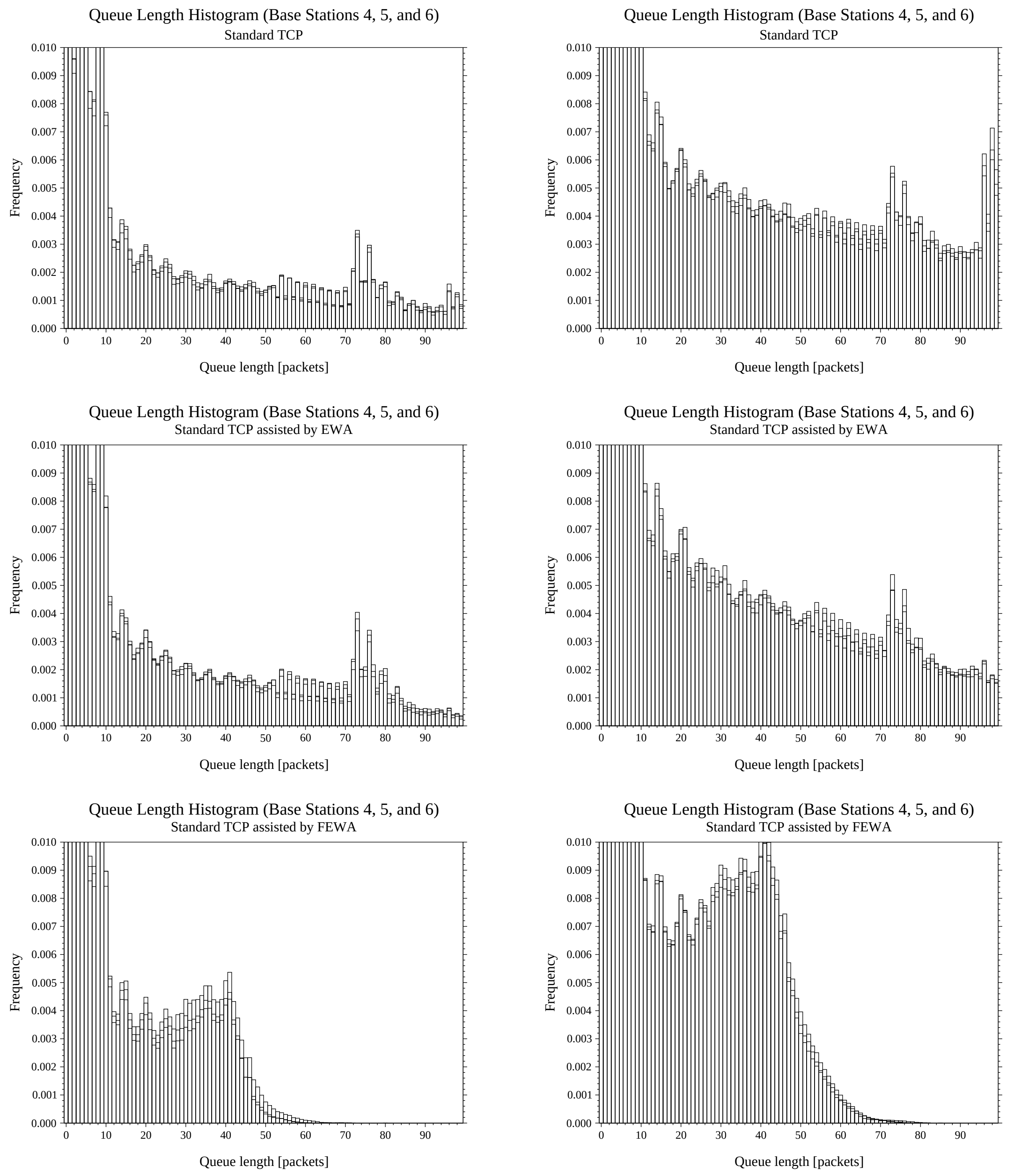


Figure 16.2: Queue length histograms in the downstream queues of the base stations in RAN 2 for standard TCP, TCP+EWA, and TCP+FEWA (lower mean network load on the left side, higher mean network load on the right side, for each of the considered approaches the histograms of the base stations 4–6 are put on top of each other)

to slightly decrease the maximum queue length in these base stations. But in the scenario with a higher mean network load, standard TCP and TCP+EWA are both not able to prevent packet losses due to congestion. For the scenario with a lower mean network load, TCP+FEWA reaches considerably lower maximum queue lengths. In the scenario with a higher mean network load TCP+FEWA considerably

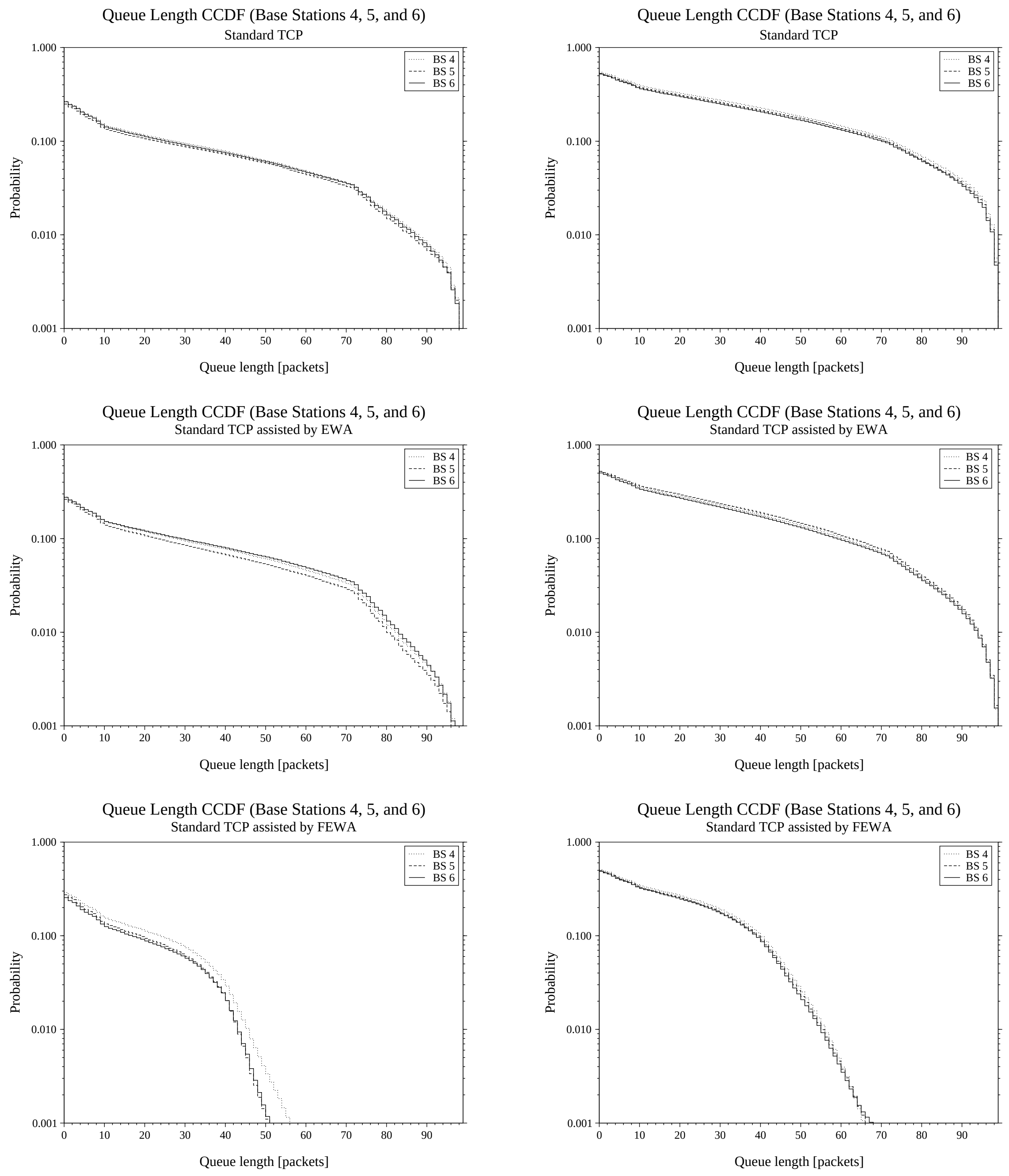


Figure 16.3: Queue length complementary cumulative distribution functions in the downstream queues of the base stations in RAN 2 for standard TCP, TCP+EWA, and TCP+FEWA (lower mean network load on the left side, higher mean network load on the right side)

reduces the frequency of very full queues. This observation is formalized by the histograms and the complementary cumulative distribution functions (CCDFs) of the queue length observations after the transient phase, shown in Figure 16.2 and Figure 16.3, respectively. The histograms depict the queue length of the downstream queues of the base stations 4–6, the base stations in RAN 2, for both simulation scenarios. To show that the obtained results are similar for the base stations 4–6 for each of the considered approaches these histograms are put on top of each other. For the same queues the CCDFs depict the probability to observe queue lengths that are above a given queue length. The y-axis representing this probability is logarithmically scaled to focus on the tail probabilities for larger queue lengths. In contrast to the other two approaches, it can be seen that with TCP+FEWA the occurrence of larger queue lengths in the considered queues is very unlikely or is even prevented.

For standard TCP, the queues in the base stations of RAN 2 have a mean queue length of 7.16 packets in the scenario with a lower mean network load and 19.53 packets in the scenario with a higher mean network load. If standard TCP is assisted by EWA, the first value is slightly increased to 7.21 packets while the second value is considerably decreased to 16.65 packets. Standard TCP assisted by FEWA reaches the lowest mean queue lengths of 4.64 packets and 11.41 packets, respectively. More importantly, TCP+FEWA prevents that packets are lost due to a congested queue. Thus, routers that are equipped with FEWA are able to handle much more TCP connections without being congested.

In the simulated network topology and with the given traffic model the base stations in the RAN 2 are lowly to moderately loaded most of the simulated time. Therefore, the utilization factor α in the EWA algorithm is increased to such high values in periods of time with a lower load in the base stations that it cannot be decreased as fast as necessary in periods of time with highly loaded or even congested base stations. Thus, the EWA algorithm is not able to adequately control the sending window of the TCP senders to avoid congestion and packet losses in the base stations if a realistic traffic model for IP-based networks is used. Only the FEWA algorithm is able to early react on impending congestion in the base stations and adequately limit the sending window of TCP senders such that no packets at all are lost due do congestion in the base stations.

16.5 Conclusion and Outlook

Using mechanisms that provide explicit congestion feedback information from routers appears very promising to improve the performance of TCP connections. One of these approaches, EWA, suffers from a major shortcoming as it cannot adequately support bursty traffic. Therefore, a new EWA-related algorithm called FEWA has been developed which adapts the main congestion-control algorithm in EWA-capable routers by a more appropriate fuzzy-based calculation. In this chapter, the performance of standard TCP and standard TCP either assisted by EWA or assisted by FEWA is investigated.

Using two simulation scenarios with different mean network loads, the following results are achieved: If routers in the network are lowly to moderately loaded in the mean that congestion is likely to happen, the throughput of TCP connections traversing these routers is increased if TCP's end-to-end congestion control is assisted by well-designed explicit congestion feedback approaches from routers. In these cases, TCP+FEWA is able to outperform standard TCP with a throughput increase of 2–4 % and TCP+EWA with a throughput increase of even 2–5 %. In addition, FEWA completely prevents packet losses due to congestion in the base stations of RAN 2. If routers in the network are not congested at all, the throughput of TCP connections traversing these routers are not impacted or only very slightly decreased compared to standard TCP if TCP's end-to-end congestion control is assisted by EWA or assisted by FEWA. Thus, EWA and FEWA only reduce the sending window of TCP senders if it is necessary in order to reduce or completely avoid losses due to (impending) congestion in routers.

Considering these results, the implementation of FEWA in routers to assist TCP’s end-to-end congestion control is highly recommended. FEWA reaches considerable performance gains with a low additional computational complexity in routers (cf. Section 13.2.2). Therefore, FEWA is another example that fuzzy-based congestion control algorithms are able to more adequately react on changing load in routers than approaches based on other control-theoretic assumptions, e.g., time series.

In addition, it might be interesting to investigate how the fuzzy controller of the FEWA algorithm can be combined with other existing approaches, e.g., RED [39] or Explicit Control Protocol (XCP) [90, 91]. In combination with RED a modified FEWA fuzzy controller can then be used to determine the packet loss probability dependent on the current queue length. And the efficiency controller of XCP can be replaced by a modified FEWA fuzzy controller to reach a higher performance of aggregated traffic over a router with XCP-capabilities. The latter approach will be investigated in more detail in an ongoing project with Ericsson Research.

Chapter 17

Performance Evaluation of ETCP and XCP by Simulations

17.1 Introduction

In the following, two simulation scenarios are considered which differ in their mean load in the network: the first simulation scenario uses $n = 24$ TCP, ETCP, or XCP senders in each LAN of the public Internet to generate WWW traffic in the network, the second one doubles this value. In each scenario the performance metrics of standard TCP are compared with the performance metrics of ETCP (assisted by FEWA) or XCP. For this comparison the ETCP variant 3 is used, since it has provided the best results in different test simulations. In addition, for each simulation scenario the queue length process, the queue length histogram, and the complementary cumulative distribution function (CCDF) of the queue length of each downstream queue in the base stations in RAN 2 are shown.

Currently, twenty simulation runs per scenario case are performed. Every run for each simulation scenario with either a lower or a higher mean network load considers a simulated time of 18000 seconds. The transient analysis of the simulations (cf. Appendix C) shows that the throughputs of TCP, ETCP, or XCP connections have a transient phase of about 1800 seconds. Therefore, only the throughput observations after the transient phase can be used to calculate the mean values of the different throughput metrics. In addition, also the mean numbers of packet losses in the downstream queues of the base stations in RAN 2 are calculated by ignoring packet losses during the transient phase.

The last column Δ in the following Tables 17.1 and 17.2 denotes whether the simulation results of one approach compared to the simulation results of another approach are statistically significantly different or not. In this column the approaches are compared in the following order: standard TCP $\leftrightarrow$ ETCP (FEWA), standard TCP $\leftrightarrow$ XCP, and ETCP (FEWA) $\leftrightarrow$ XCP. A '-' denotes that the first approach of the comparison reaches significantly better results while a '+' denotes that the second approach reaches significantly better results. And a '=' means that with the simulation results no significant difference between the two compared approaches can be concluded. The details of the statistical evaluation of these simulation results can be found in Appendix D.

17.2 Scenario 1: TCP NewReno, ETCP (FEWA), and XCP, Lower Mean Network Load

Table 17.1: Simulation results of scenario 1

		standard TCP	ETCP (FEWA)	XCP	Δ
$\overline{TPO}$ of TCP connections [segments/second]	(W)LAN 1	52.77	58.88	54.18	$++-$
	(W)LAN 2	52.78	59.10	53.87	$++-$
	(W)LAN 3	53.04	59.00	54.10	$++-$
	(W)LAN 4	37.78	43.50	34.80	$+--$
	(W)LAN 5	37.72	43.56	34.87	$+--$
	(W)LAN 6	37.82	43.60	34.83	$+--$
$\overline{TPC}$ of TCP connections [segments/second]	(W)LAN 1	47.59	58.61	51.17	$++-$
	(W)LAN 2	47.65	58.67	51.07	$++-$
	(W)LAN 3	47.71	58.60	51.13	$++-$
	(W)LAN 4	36.36	42.49	32.14	$+--$
	(W)LAN 5	36.36	42.57	32.09	$+--$
	(W)LAN 6	36.36	42.60	32.12	$+--$
mean number of packet losses in base station ((W)LAN router)	(W)LAN 1	0.00	0.00	0.00	$===$
	(W)LAN 2	0.00	0.00	0.00	$===$
	(W)LAN 3	0.00	0.00	0.00	$===$
	(W)LAN 4	2581.10	0.00	0.00	$++=$
	(W)LAN 5	2598.30	0.00	0.00	$++=$
	(W)LAN 6	2561.50	0.00	0.00	$++=$

In this scenario with a lower mean network load the base stations in RAN 1 are not congested at all, i.e., no packets are lost in the downstream queues of these base stations, and the base stations in RAN 2 are rarely congested if standard TCP connections are considered.

The throughputs of standard TCP connections traversing the base stations in RAN 1 are the lowest compared to the other two approaches. ETCP connections reach an approximately 12 % higher overall mean throughput and an approximately 23 % higher connection-oriented mean throughput. And XCP connections increase the overall and connection-oriented throughput of standard TCP connections by 2 % and 7 %.

In contrast, the throughputs of standard TCP connections traversing the base stations in RAN 2 are not the lowest compared to the other two approaches. ETCP connections reach higher throughputs (15 % and 17 % gain for the overall and connection-oriented mean throughput), but XCP connection reach lower throughputs (8 % and 13 % loss for the overall and connection-oriented mean throughput) compared to standard TCP.

And compared to standard TCP packet losses can be completely prevented by ETCP and by XCP.

17.3 Scenario 2: TCP NewReno, ETCP (FEWA), and XCP, Higher Mean Network Load

Table 17.2: Simulation results of scenario 2

		standard TCP	ETCP (FEWA)	XCP	Δ
$\overline{TPO}$ of TCP connections [segments/second]	(W)LAN 1	47.72	47.46	46.89	$=-=$
	(W)LAN 2	47.72	47.91	46.91	$=--$
	(W)LAN 3	47.97	47.70	46.74	$=-=$
	(W)LAN 4	28.31	32.00	25.87	$+--$
	(W)LAN 5	28.10	31.73	25.86	$+--$
	(W)LAN 6	28.17	32.27	25.96	$+--$
$\overline{TPC}$ of TCP connections [segments/second]	(W)LAN 1	44.98	54.16	45.82	$++-$
	(W)LAN 2	44.94	54.24	45.81	$++-$
	(W)LAN 3	45.06	54.48	45.86	$++-$
	(W)LAN 4	30.46	34.92	26.05	$+--$
	(W)LAN 5	30.42	34.81	26.07	$+--$
	(W)LAN 6	30.35	35.10	26.08	$+--$
mean number of packet losses in base station ((W)LAN router)	(W)LAN 1	0.00	0.00	0.00	$===$
	(W)LAN 2	0.00	0.00	0.00	$===$
	(W)LAN 3	0.00	0.00	0.00	$===$
	(W)LAN 4	14840.70	0.00	63.75	$++=$
	(W)LAN 5	15196.55	0.00	0.00	$++=$
	(W)LAN 6	14980.55	0.00	58.80	$++=$

In this scenario with a higher mean network load the base stations in RAN 1 are still not congested at all, i.e., no packets are lost in their downstream queues, and the base stations in RAN 2 are much more often congested than in the previous scenario if standard TCP connections are considered. Standard TCP connections and ETCP connections traversing the uncongested base stations reach nearly the same overall mean throughput, XCP connections are slightly worse (approximately 2 % loss). The connection-oriented mean throughput of ETCP connections is approximately 18 % and 21 % higher than the connection-oriented mean throughput of XCP and standard TCP connections. The reason for this large discrepancy between overall and connection-oriented mean throughput of ETCP connections compared to the other two approaches is that the ETCP senders are allowed to send much more segments per time interval into the network than the other senders. This causes a higher collision probability in the

LANs with the effect that some segments are delayed and even discarded by the LAN medium access control (MAC) if their maximum number of retransmissions (in Ethernet: 16) is exceeded. The few ETCP connections affected by such delayed or discarded segments reach a much lower throughput than other ETCP connections. And that decreases the overall mean throughput of ETCP connections but has only a slight effect on the connection-oriented mean throughput.

For connections traversing the base stations in RAN 2 the same qualitative result than in the scenario with a lower mean network load can be observed. ETCP connections reach the highest overall and connection-oriented mean throughput. They outperform standard TCP connections by approximately 14 % and 15 % and XCP connections by approximately 24 % and 34 %, respectively.

Compared to standard TCP packet losses in the base stations of RAN 2 can be completely prevented or largely reduced if ETCP or XCP is used. A finer evaluation of the XCP simulations shows that only in a single simulation run packet losses can be observed. Thus, in nearly all cases packet losses can be also prevented if standard TCP is replaced by XCP.

17.4 Queue Lengths in Bottleneck Base Stations

In this section, the queues of the base stations in RAN 2 are considered in more detail. Figure 17.1 shows typical queue length processes in the downstream queue of a base station in RAN 2 for both simulation scenarios and for an observation interval of 120 s.

In both scenarios with a lower and a higher mean network load and compared to standard TCP, ETCP is able to slightly decrease the maximum queue length in these base stations. And XCP is able to largely decrease the maximum queue length. This observation is formalized by the histograms and the complementary cumulative distribution functions (CCDFs) of the queue length observations after the transient phase, shown in Figure 17.2 and Figure 17.3, respectively. The histograms depict the queue length of the downstream queues of the base stations 4–6, the base stations in RAN 2, for both simulation scenarios. To show that the obtained results are similar for the base stations 4–6 for each of the considered approaches these histograms are put on top of each other. For the same queues the CCDFs depict the probability to observe queue lengths that are above a given queue length. The y-axis representing this probability is logarithmically scaled to focus on the tail probabilities for larger queue lengths. In contrast to standard TCP, it can be seen that with ETCP and XCP the occurrence of larger queue lengths in the considered queues is very unlikely or is even prevented.

For standard TCP, the queues in the base stations of RAN 2 have a mean queue length of 7.16 packets in the scenario with a lower mean network load and 19.53 packets in the scenario with a higher mean network load. If ETCP is used, the first value is slightly decreased to 5.37 packets while the second value is considerably decreased to 13.57 packets. XCP reaches the lowest mean queue lengths of 0.77 packets and 1.60 packets, respectively. More importantly, XCP prevents (in nearly all cases) that packets are lost due to a congested queue. Thus, routers that are equipped with XCP are able to handle much more XCP connections (on a lower throughput level) without being congested compared to all other approaches.

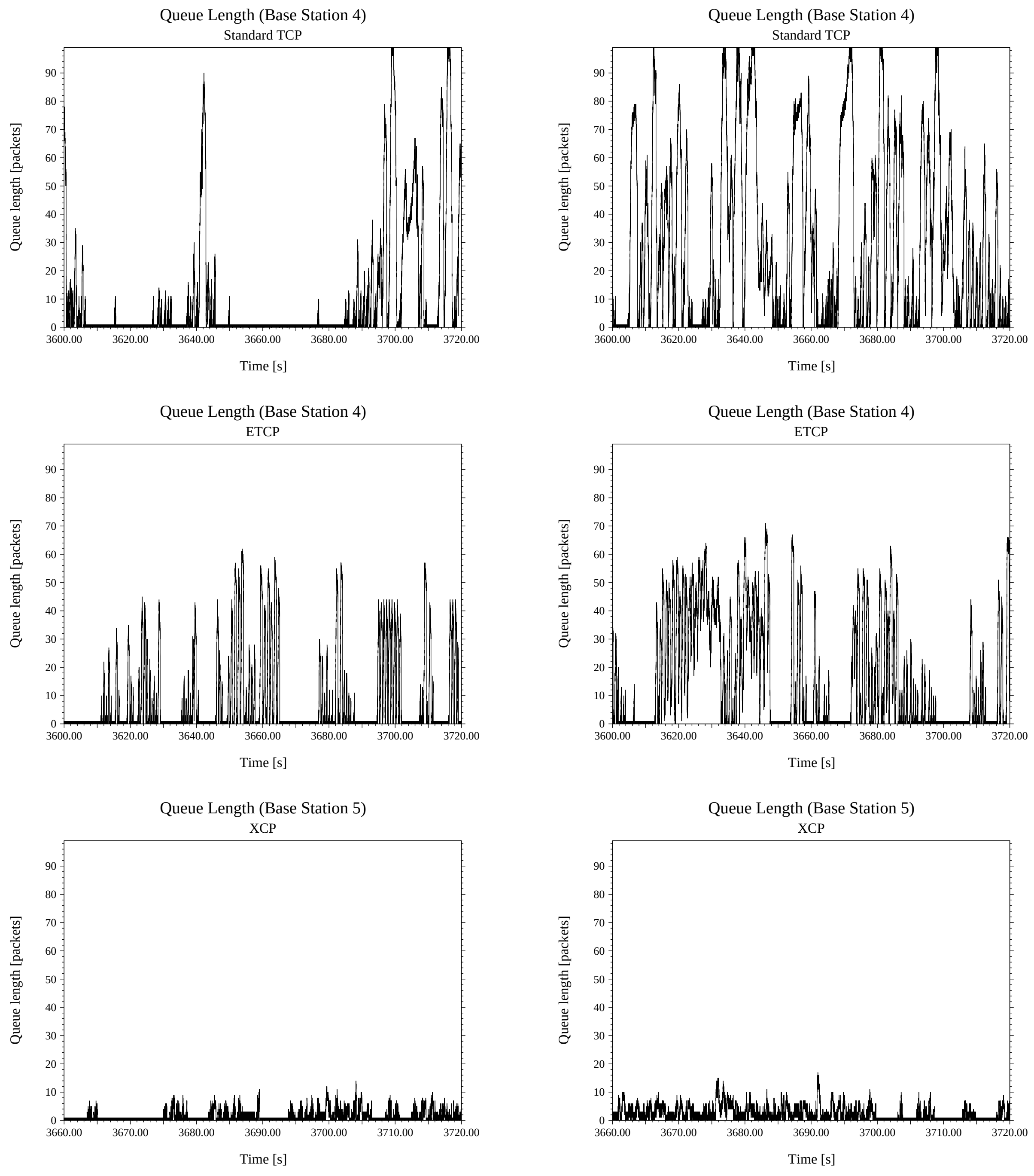


Figure 17.1: Queue length process in the downstream queue of a base station in RAN 2 for standard TCP, ETCP, and XCP (lower mean network load on the left side, higher mean network load on the right side)

17.5 Conclusion and Outlook

Two load scenarios are used to investigate the performance of standard TCP compared to more sophisticated RCF approaches ETCP and XCP than those considered in the last chapter. The following results are achieved: If routers in the network are lowly to moderately loaded in the mean that congestion is likely to happen, ETCP reaches a 15 %-14 % higher throughput than standard TCP and a 23 %-24 % higher throughput than XCP if the overall mean throughput is considered. The gain of ETCP regarding

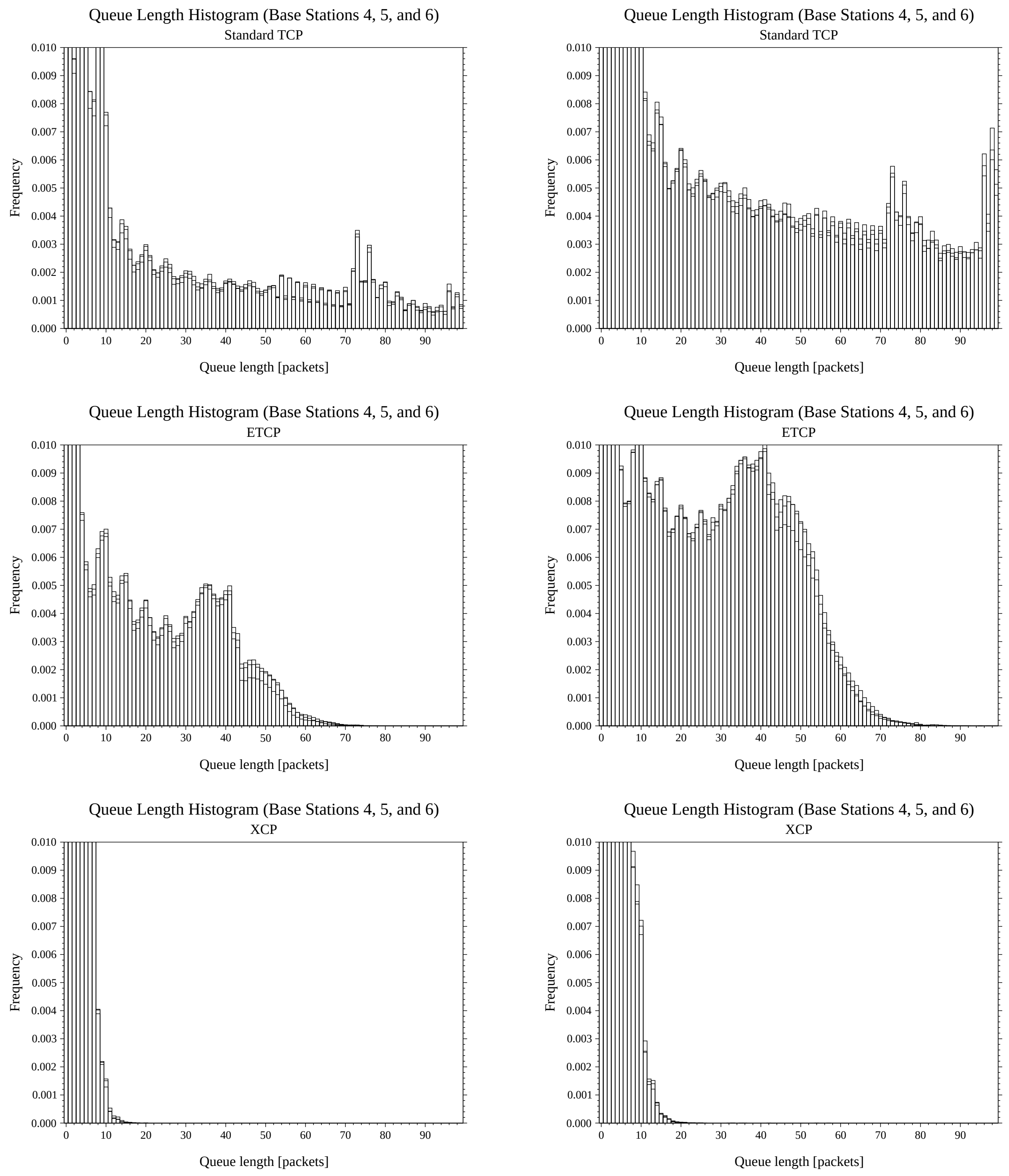


Figure 17.2: Queue length histograms in the downstream queues of the base stations in RAN 2 for standard TCP, ETCP, and XCP (lower mean network load on the left side, higher mean network load on the right side, for each of the considered approaches the histograms of the base stations 4–6 are put on top of each other)

the connection-oriented mean throughput is even (slightly) higher. In addition and similar to XCP, ETCP is able to prevent packet losses in such routers while standard TCP loses thousands of packets. In other words, XCP reaches the worst throughput results for connections traversing such loaded routers compared to the other two approaches and even compared to TCP+EWA and TCP+FEWA (cf. Chapter 16).

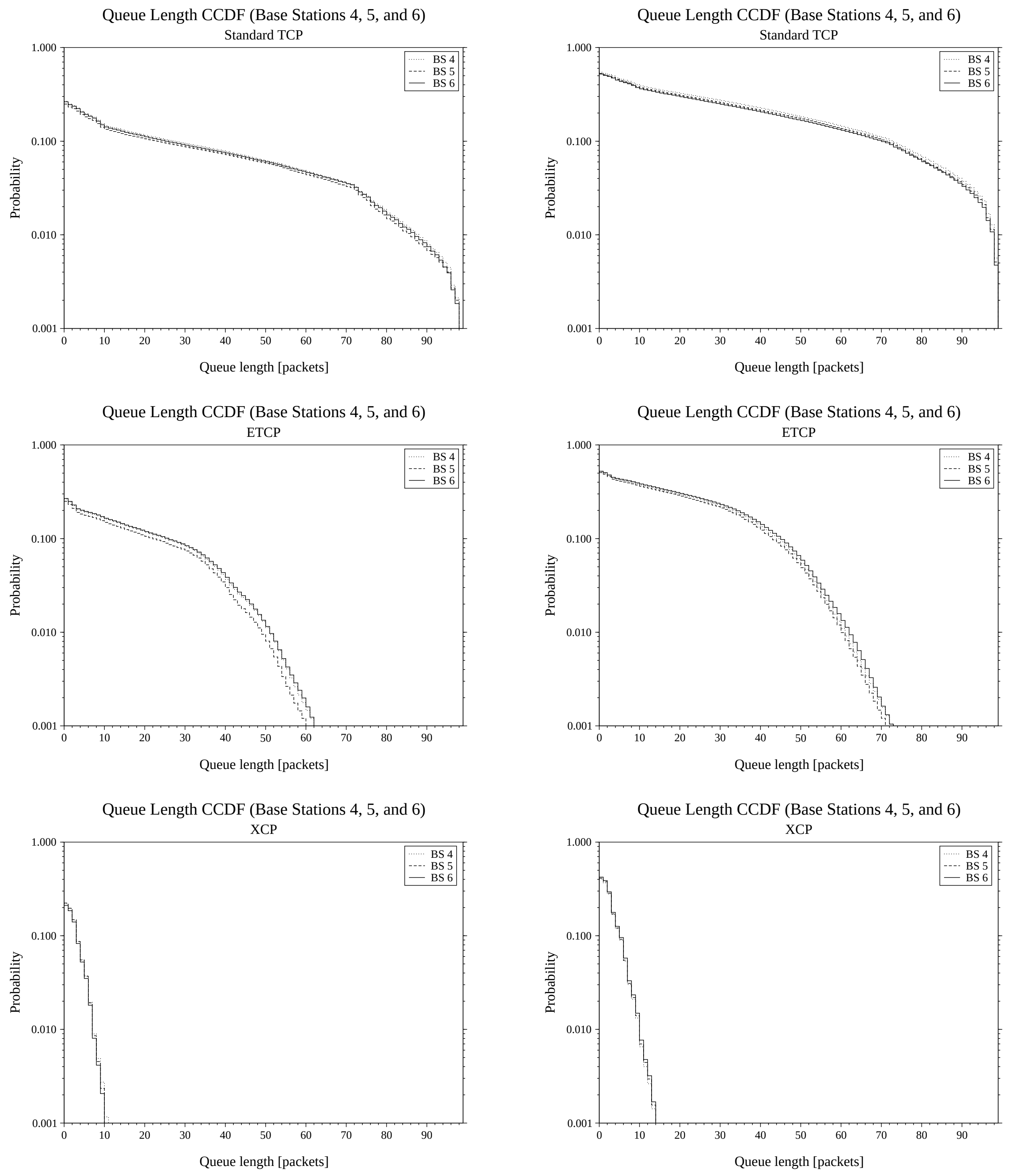


Figure 17.3: Queue length complementary cumulative distribution functions in the downstream queues of the base stations in RAN 2 for standard TCP, ETCP, and XCP (lower mean network load on the left side, higher mean network load on the right side)

This is the outcome of the efficiency controller of XCP which does not adequately work in this case and which should be replaced by a new controller, e.g., based on modified FEWA control algorithms. But in its current version, XCP should not be used in routers in front of links with a comparatively low bandwidth, e.g., a low-speed (W)LAN or other (wireless) access technologies. This has been shown by other simulations using UMTS as a wireless access technology [105]. The only advantage XCP has compared to all other approaches is that XCP is able to keep the queue length short in these routers with the result that the round trip time of the network path seen by an XCP sender is much lower compared to the round trip time seen by a TCP or an ETCP sender. This might be beneficial for applications that require a shorter response time.

Even if routers in the network are not congested at all, ETCP is the best approach. The expected throughput of a single ETCP connection is much higher than the expected throughput of a standard TCP or an XCP connection. Here, the throughput gain of ETCP is 18 % and 21 % compared to XCP and standard TCP. This advantage of ETCP is combined with a much higher load in the LANs which leads to an increased probability that packets in each LAN collide. As a result, packets from ETCP connections are more delayed or even more often discarded by the MAC of a LAN than packets of other connections. That is the reason why the overall mean throughput of ETCP connections is not or not so largely increased compared to standard TCP or XCP connections and dependent on the load in the routers.

Considering these results, it is highly recommended to implement the ETCP algorithms in a TCP sender to better take advantage of network-load information obtained by FEWA-capable routers. Then, with only a few additional operations the throughput of a standard TCP sender can be (largely) increased without overloading the network.

Chapter 18

Congestion Feedback from Routers: Conclusion and Outlook

The second part of this dissertation has considered NIS approaches between routers and end systems based on congestion feedback from routers. These NIS approaches can be classified in one- or two-way RCF approaches and transparent or non-transparent RCF approaches for TCP instances in the end systems. Most of the existing RCF approaches have been identified as too limited in their applicability, too complex, or too restricted in their (expected) performance gain compared to standard TCP. Therefore, two new RCF approaches have been developed. The first new approach, called FEWA, is a one-way RCF approach transparent for TCP instances in the end systems. It is an improvement of the existing RCF approach EWA. The second new approach, called ETCP, is also a one-way RCF approach but it is non-transparent for TCP senders, since it requires a few changes in the algorithms of a TCP sender. ETCP can be used in combination with any RCF approach that uses the advertised receiver window in TCP acknowledgments to inform the TCP senders about the current load in the network. But in this dissertation, ETCP is investigated in combination with FEWA, since FEWA is the best existing approach of this kind.

These two new RCF approaches and two existing RCF approaches, called EWA and XCP, are implemented in a simulation environment to investigate their performance by simulations. Transparent RCF approaches like EWA and FEWA and non-transparent RCF approaches like ETCP and XCP are separately compared to standard TCP.

For transparent RCF approaches, FEWA reaches the highest throughput of all considered approaches. In addition, FEWA is able to completely prevent packet losses in routers while the other two approaches loses thousands of packets. Thus, if a transparent RCF approach is required, FEWA should be used as the first choice. But with sophisticated developed non-transparent RCF approaches a higher performance compared to FEWA should be reached.

Regarding non-transparent RCF approaches, ETCP reaches the highest throughput of all considered approaches. In addition, packet losses are completely prevented by ETCP XCP is able to nearly prevent packet losses but at the cost of a (slightly) reduced throughput compared to ETCP. It is remarkable that in some cases the throughput reached by XCP is even lower than the throughput of EWA, FEWA, or standard TCP. Thus, XCP should not be used in routers connected to links with a comparatively low bandwidth, e.g., gateway routers to low-speed (W)LANs or other (wireless) access technologies.

It is very interesting that a well-designed one-way RCF approach like ETCP is able to improve the performance reached with the supposed well-designed two-way RCF approach XCP. Since the algorithms performed in an XCP sender are well-designed, one of the controllers performed in an XCP-capable router, the efficiency controller, seems to be improvable. For example, the efficiency controller of XCP should be replaced by a modified FEWA controller to use XCP for a wider application area, e.g., in WCDMA-based wireless access technologies like UMTS with largely varying bandwidths. This is investigated in an ongoing project with Ericsson Research.

Chapter 19

Network-Information Sharing in IP-based Networks: Conclusion and Outlook

In this dissertation, the functionalities, properties, applicability, and performance of several NIS approaches for TCP connections are considered. Some of these NIS approaches, NIS approaches of the first type, are located in a single Internet end system and share available network information between the senders of some of its TCP connections. This network-information sharing can be done either once to start the sender of a new TCP connection with more adequate initial values for some of its control variables or it can be done in a continuous way during the whole lifetime of TCP connections to always adjust the control variables of their senders on the newest available network information. The former NIS approaches are called one-time network-information sharing approaches while the latter NIS approaches are called common congestion control approaches. Other NIS approaches, NIS approaches of the second type, are located in Internet routers and Internet end systems that network information collected in the routers can be used to determine congestion-feedback information that is sent back to the end systems to better control the sending window of the senders of all TCP connections in these end systems. Such RCF approaches can be either transparent or non-transparent for the TCP instances running in the end systems. In addition, they can be also based on a one-way or two-way information exchange between the routers and the TCP senders located in the end systems.

For the first type of NIS approaches simulations have shown that one-time network-information sharing approaches like the new FS-TCBI are able to improve the throughput of new TCP connections entering an ensemble. But this is in most cases combined with a fairness and throughput decrease of the whole ensemble. The reason for this result is that such NIS approaches are not able to keep the best-case property of TCP, i.e., to permanently equally share the available bandwidth, over the whole lifetime of the TCP connections in an ensemble. This can be only reached by the new common congestion-control approach EFCM. Its imitation of TCP's best-case behavior has been analytically shown and results in a (largely) improved fairness in all cases and throughput in most cases, evaluated by simulations and measurements. As a result, NIS approaches of this type—and in particular EFCM—are able to reach a (large) performance gain for some TCP connections of an end system. What is yet an open question is how it can be generally found out by an Internet end system which of its TCP connections can form an ensemble that their senders can gainfully share network information. Thus, implementing such NIS approaches in the real Internet that can cooperate with existing IP-based mechanisms like Mobile IP or NAT is still a practical challenge.

Since NIS approaches of the second type do not have the inherent applicability obstacle of the NIS approaches of the first type, they can be much easier implemented in the real Internet. But in contrast to the first-type NIS approaches this can be only promoted by the network providers, it is out of the feasibility of the end-system users to implement such RCF approaches. Nevertheless, these NIS approaches are able to achieve slight throughput gains if they are transparent and much larger throughput gains if they are non-transparent for the TCP instances running in the end systems. In the simulations, the new one-way RCF approaches FEWA and ETCP outperform the existing ones regardless if they are one- or two-way RCF approaches. As a result, NIS approaches of this type—and in particular FEWA and ETCP—are able to reach a slight to medium throughput gain for all TCP connections of a network which traverse bottleneck routers equipped with RCF capabilities.

NIS approaches of the second type can be used to assist NIS approaches of the first type to decide which TCP connections can form an ensemble. And NIS approaches of the first type can be used to distribute network information obtained by NIS approaches of the second type among several TCP connections. But NIS approaches of the second type can even render unnecessary some NIS approaches of the first type, since in combination with a promising RCF approach like FEWA or ETCP the (E)TCP senders are sufficiently well informed by the network about the current load conditions in the network. It might be advantageous to combine FEWA-assisted TCP or ETCP with one-time network-information sharing approaches to faster adjust the congestion-control variables of a new (E)TCP connection entering an ensemble on the current network conditions provided by the RCF approach. A common congestion controller like EFCM can further but in general only slightly improve the performance of such ensemble- or temporal-TCBI controlled (E)TCP connections, since jointly controlled (E)TCP senders are then able to react only a little bit faster on current congestion feedback from the routers. But this slight additional performance gain of EFCM in combination with FEWA and/or ETCP is rather low compared to the complexity such a hybrid approach will introduce in the end systems.

To comprise, the implementation of FEWA in Internet routers and—as a promising extension—ETCP in Internet end systems is recommended for two reasons: (1) to increase the throughput of (E)TCP connections and (2) to significantly reduce the probability of congestion in routers or even to prevent congested routers at all in the network. Both network users and network providers similarly benefit from these two RCF approaches. FEWA and ETCP have a low complexity and require no or at most only a few changes in the senders of TCP connections (if symmetrical routing cannot be guaranteed in the whole network—or at least in the bottleneck routers—both approaches require also slight changes in the receivers of TCP connections). And compared to their low complexity their performance gain is remarkable.

In an ongoing project with Ericsson Research RCF approaches designed for routers which are located in specific access networks, e.g., UMTS, are considered. Since these routers are aware of the numbers of TCP connections traversing them and know the available bandwidth in the succeeding (wireless) link, they are able to calculate more appropriate congestion-feedback information which can be used by the senders of the TCP connections to better adjust their sending window on this network information to achieve a higher throughput.

Appendix

Appendix A

Transient Analysis Part I

A.1 Transient Analysis Method

The method used for the transient analysis of the simulation results is called initial data deletion [1]. Suppose there are m replications with n_i observations in replication i, $1 \leq i \leq m$. The jth observation in the ith replication is denoted x_{ij}. Let n be the minimum of all n_i. Then the method works as follows:

(1) By averaging across the replications a mean value $\overline{x}_j$ for the jth observation is calculated:

$$\overline{x}_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij}, \quad j = 1, 2, \ldots, n \tag{A.1}$$

(2) Using the values from (1) an overall mean value $\overline{\overline{x}}$ is calculated:

$$\overline{\overline{x}} = \frac{1}{n}\sum_{j=1}^{n} \overline{x}_j \tag{A.2}$$

(3) Vary l between 1 and $n-1$:

(a) Delete the first l observations from the simulation results and calculate an overall mean value $\overline{\overline{x}}_l$ for the remaining $n-l$ observations:

$$\overline{\overline{x}}_l = \frac{1}{n-l}\sum_{j=l+1}^{n} \overline{x}_j \tag{A.3}$$

(b) Compute the relative change Δ_l in the overall mean value:

$$\Delta_l = \frac{\overline{\overline{x}}_l - \overline{\overline{x}}}{\overline{\overline{x}}} \tag{A.4}$$

A plot of Δ_l as a function of l shows the transient phase of the simulation results. If the plot stabilizes at a point l^*, then the first l^* observations of the simulations reflect the transient phase of the simulation results and should not be considered for the mean calculation.

A.2 Transient Analysis Results

As an example, the following Figure A.1 shows the transient analysis for the throughput of TCP connections traversing a network path with a minimum round trip time of 100 ms and a reliable last hop. Figure A.2 shows the same l-Δ_l plot in more detail.

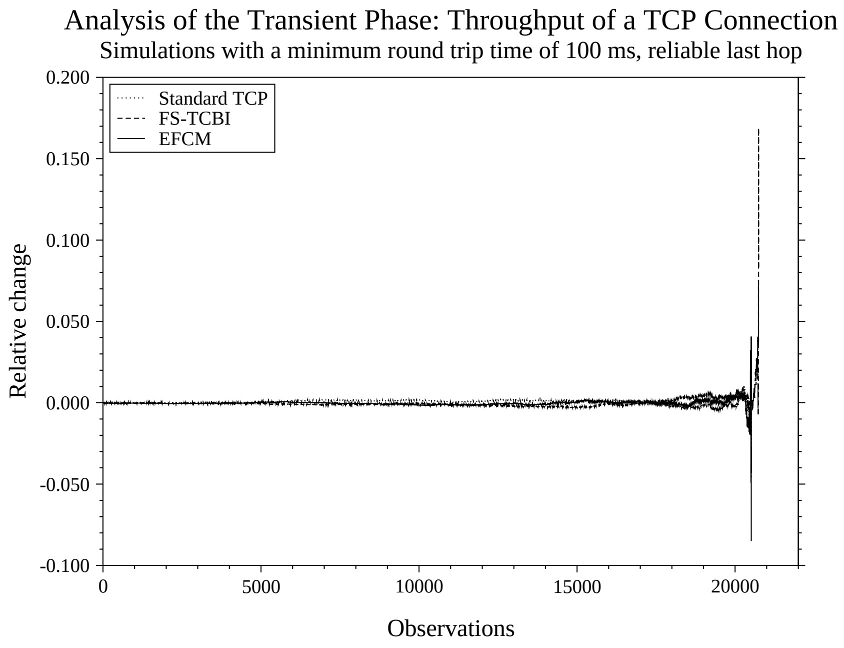


Figure A.1: Transient analysis for the simulations of Part I with a minimum round trip time of 100 ms and a reliable last hop

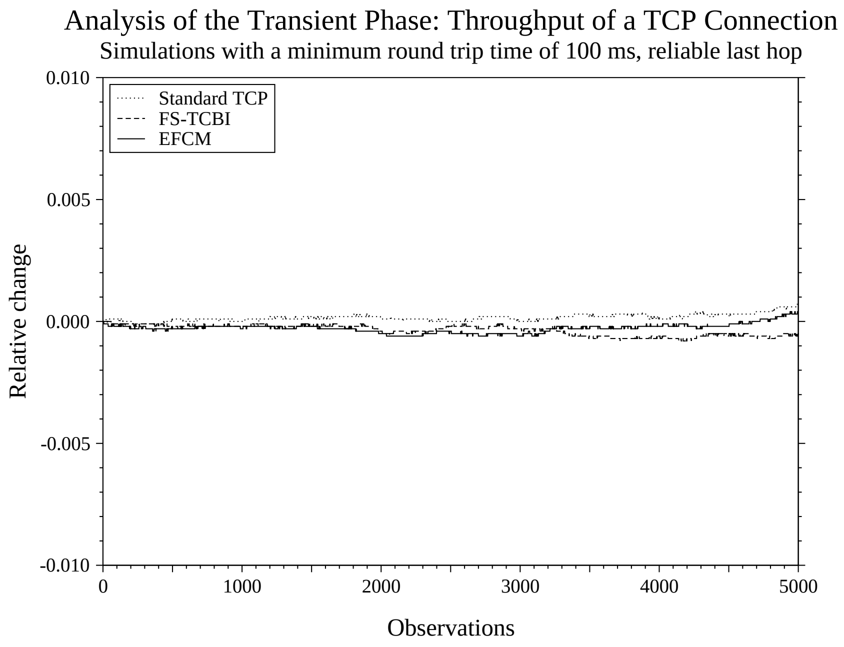


Figure A.2: Zoomed transient analysis for the simulations of Part I with a minimum round trip time of 100 ms and a reliable last hop

Another example pictured in Figure A.3 shows the transient analysis for the throughput of TCP connections traversing a network path with a minimum round trip time of 100 ms and an unreliable last hop. Figure A.4 shows the same l-Δ_l plot in more detail.

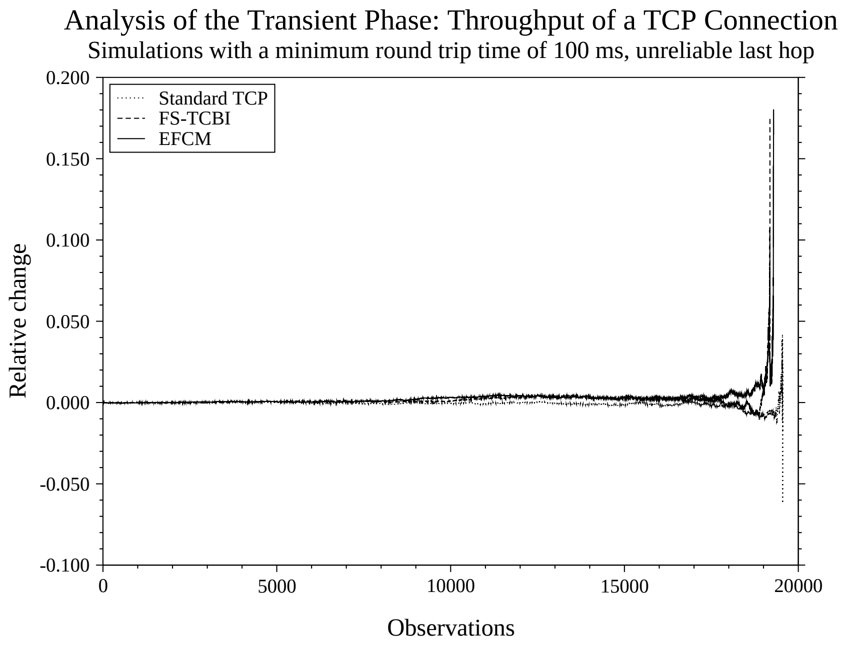


Figure A.3: Transient analysis for the simulations of Part I with a minimum round trip time of 100 ms and an unreliable last hop

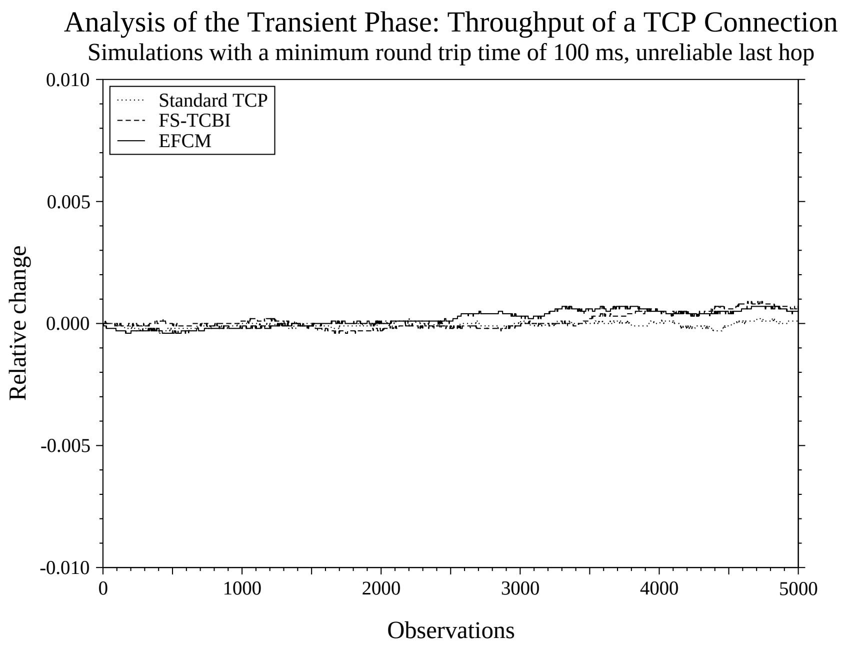


Figure A.4: Zoomed transient analysis for the simulations of Part I with a minimum round trip time of 100 ms and an unreliable last hop

Considering these four figures and taking into account that transient analysis for other simulation scenarios show a similar result, it can be assumed that the simulation results of Part I have no transient phase for the throughput of TCP connections. Therefore, the mean values of the performance metrics are calculated by using all observed TCP throughputs during the simulated time.

Appendix B

Statistical Evaluation Part I

B.1 Statistical Evaluation Method

The statistical evaluation method used for this comparison is called the t-test for unpaired observations of two alternatives and is described in detail in [1]. The main idea of this method is to compute a confidence interval for the difference of the mean values of both alternatives for a given confidence level. Then the decision criterion is:

- If the confidence interval includes zero, then the two alternatives cannot be distinguished.
- If the confidence interval is above/below zero, then the first/second alternative is the better one.

Tests with confidence intervals give not only a yes-no answer like other hypothesis tests, they also give an answer to the question how precise the decision is. A narrow confidence interval indicates that the precision of the decision is high whereas a wide confidence interval indicates that the precision of the decision is rather low.

This t-test for unpaired observations of two alternatives is used for the statistical evaluation of the simulation results for both the overall mean throughput ($\overline{TPO}$) and the connection-oriented mean throughput ($\overline{TPC}$) of new or concurrent TCP connections controlled by either the standard TCP, the FS-TCBI, or the EFCM controller, respectively, and the mean fairness index ($\overline{I}_f$) of concurrent TCP connections.

The values for this statistical evaluation are produced by twelve independent simulation runs for every last hop scenario in the simulation model.

B.2 Statistical Evaluation Results for FS-TCBI Simulations

In the following Tables B.1 to B.8 the statistical evaluation of the simulation results are shown. For each confidence interval also the confidence level (0.90, 0.95 or 0.99) is depicted. If the simulation results for the standard TCP and the FS-TCBI controller are not significantly different even for the confidence level 0.90, then the confidence interval for the confidence level 0.90 is stated.

Table B.1: Statistical evaluation of the FS-TCBI simulation results of simulation 1, scenario 1

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 7.31, -\ 2.03)$
	TCP 2	$0.99 : (-\ 6.35, -\ 0.82)$
	TCP 3	$0.99 : (-\ 7.29, -\ 2.22)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 2.96, -\ 0.13)$
	TCP 2	$0.95 : (-\ 2.19, -\ 0.19)$
	TCP 3	$0.99 : (-\ 2.82, -\ 0.20)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+\ 0.43, +\ 2.14)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (+\ 0.31, +\ 1.90)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.90 : (-\ 0.00, +\ 0.01)$

Table B.2: Statistical evaluation of the FS-TCBI simulation results of simulation 1, scenario 2

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 13.73, -\ 10.16)$
	TCP 2	$0.99 : (-\ 13.92, -\ 9.44)$
	TCP 3	$0.99 : (-\ 13.67, -\ 9.59)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.90 : (-\ 0.41, +\ 1.93)$
	TCP 2	$0.95 : (+\ 0.17, +\ 3.46)$
	TCP 3	$0.95 : (+\ 0.00, +\ 3.08)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 5.52, -\ 2.77)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (+\ 1.97, +\ 4.73)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.90 : (-\ 0.00, +\ 0.01)$

Table B.3: Statistical evaluation of the FS-TCBI simulation results of simulation 2, scenario 1

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 5.20, - \quad 2.84)$
	TCP 2	$0.99 : (- \quad 5.59, - \quad 1.63)$
	TCP 3	$0.99 : (- \quad 5.99, - \quad 2.90)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 3.10, - \quad 1.73)$
	TCP 2	$0.99 : (- \quad 2.60, - \quad 1.12)$
	TCP 3	$0.99 : (- \quad 2.82, - \quad 1.15)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.90 : (- \quad 0.07, + \quad 0.87)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.95 : (+ \quad 0.09, + \quad 0.84)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+ \quad 0.01, + \quad 0.02)$

Table B.4: Statistical evaluation of the FS-TCBI simulation results of simulation 2, scenario 2

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 9.88, - \quad 6.91)$
	TCP 2	$0.99 : (- \quad 8.63, - \quad 6.49)$
	TCP 3	$0.99 : (- \ 10.34, - \quad 6.68)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.90 : (- \quad 0.22, + \quad 0.86)$
	TCP 2	$0.90 : (- \quad 0.11, + \quad 0.91)$
	TCP 3	$0.90 : (- \quad 0.65, + \quad 0.77)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (- \quad 3.55, - \quad 1.94)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (+ \quad 0.28, + \quad 1.40)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+ \quad 0.00, + \quad 0.01)$

Table B.5: Statistical evaluation of the FS-TCBI simulation results of simulation 3, scenario 1

		no FS-TCBI $\leftrightarrow$ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 5.09, -\ 1.88)$
	TCP 2	$0.99 : (-\ 4.69, -\ 2.13)$
	TCP 3	$0.99 : (-\ 5.66, -\ 2.11)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 3.88, -\ 2.23)$
	TCP 2	$0.99 : (-\ 3.80, -\ 2.31)$
	TCP 3	$0.99 : (-\ 3.90, -\ 2.64)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.90 : (-\ 0.75, +\ 0.11)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 1.03, -\ 0.20)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+\ 0.00, +\ 0.01)$

Table B.6: Statistical evaluation of the FS-TCBI simulation results of simulation 3, scenario 2

		no FS-TCBI $\leftrightarrow$ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 6.64, -\ 4.76)$
	TCP 2	$0.99 : (-\ 7.09, -\ 4.97)$
	TCP 3	$0.99 : (-\ 6.45, -\ 5.59)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 1.58, -\ 0.29)$
	TCP 2	$0.99 : (-\ 1.69, -\ 0.65)$
	TCP 3	$0.99 : (-\ 1.41, -\ 0.29)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 2.22, -\ 1.40)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.95 : (-\ 0.52, -\ 0.04)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+\ 0.00, +\ 0.01)$

Table B.7: Statistical evaluation of the FS-TCBI simulation results of simulation 4, scenario 1

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 2.11, - \quad 1.25)$
	TCP 2	$0.99 : (- \quad 2.53, - \quad 1.63)$
	TCP 3	$0.99 : (- \quad 2.27, - \quad 1.35)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 1.90, - \quad 1.44)$
	TCP 2	$0.99 : (- \quad 2.14, - \quad 1.62)$
	TCP 3	$0.99 : (- \quad 1.98, - \quad 1.50)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (- \quad 0.45, - \quad 0.01)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (- \quad 0.55, - \quad 0.20)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+ \quad 0.01, + \quad 0.01)$

Table B.8: Statistical evaluation of the FS-TCBI simulation results of simulation 4, scenario 2

		no FS-TCBI ↔ FS-TCBI
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 1.40, - \quad 1.08)$
	TCP 2	$0.99 : (- \quad 1.33, - \quad 1.09)$
	TCP 3	$0.99 : (- \quad 1.30, - \quad 1.08)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (- \quad 0.64, - \quad 0.41)$
	TCP 2	$0.99 : (- \quad 0.62, - \quad 0.40)$
	TCP 3	$0.99 : (- \quad 0.53, - \quad 0.36)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (- \quad 0.73, - \quad 0.59)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (- \quad 0.53, - \quad 0.40)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (+ \quad 0.01, + \quad 0.01)$

B.3 Statistical Evaluation Results for EFCM Simulations

In the following Tables B.9 to B.16 the statistical evaluation of the simulation results are shown. For each confidence interval also the confidence level (0.90, 0.95 or 0.99) is depicted. If the simulation results for the standard TCP and the EFCM controller are not significantly different even for the confidence level 0.90, then the confidence interval for the confidence level 0.90 is stated.

Table B.9: Statistical evaluation of the EFCM simulation results of simulation 1, scenario 1

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.90 : (-\ 0.73, +\ 2.28)$
	TCP 2	$0.95 : (+\ 0.45, +\ 4.39)$
	TCP 3	$0.95 : (+\ 0.01, +\ 3.49)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 5.77, -\ 2.37)$
	TCP 2	$0.99 : (-\ 4.64, -\ 1.77)$
	TCP 3	$0.99 : (-\ 4.95, -\ 1.70)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.95 : (+\ 0.25, +\ 1.99)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 3.79, -\ 1.94)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.05, -\ 0.03)$

Table B.10: Statistical evaluation of the EFCM simulation results of simulation 1, scenario 2

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 19.76, -\ 15.84)$
	TCP 2	$0.99 : (-\ 18.90, -\ 13.90)$
	TCP 3	$0.99 : (-\ 17.37, -\ 13.04)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.90 : (-\ 1.49, +\ 1.03)$
	TCP 2	$0.90 : (+\ 0.27, +\ 3.15)$
	TCP 3	$0.90 : (+\ 0.17, +\ 2.73)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 10.35, -\ 7.67)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (+\ 1.01, +\ 3.82)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.05, -\ 0.03)$

Table B.11: Statistical evaluation of the EFCM simulation results of simulation 2, scenario 1

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.90 : (-\ 2.31, +\ 0.05)$
	TCP 2	$0.90 : (-\ 2.10, +\ 0.20)$
	TCP 3	$0.90 : (-\ 1.76, +\ 0.16)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 7.18, -\ 5.28)$
	TCP 2	$0.99 : (-\ 6.89, -\ 5.06)$
	TCP 3	$0.99 : (-\ 6.63, -\ 4.90)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.90 : (-\ 0.61, +\ 0.76)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 3.72, -\ 2.39)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.05, -\ 0.03)$

Table B.12: Statistical evaluation of the EFCM simulation results of simulation 2, scenario 2

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 16.59, -\ 13.58)$
	TCP 2	$0.99 : (-\ 14.84, -\ 12.77)$
	TCP 3	$0.99 : (-\ 17.54, -\ 13.29)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 7.70, -\ 6.08)$
	TCP 2	$0.99 : (-\ 7.34, -\ 5.56)$
	TCP 3	$0.99 : (-\ 8.94, -\ 6.30)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 9.83, -\ 8.31)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 5.82, -\ 4.42)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.04, -\ 0.03)$

Table B.13: Statistical evaluation of the EFCM simulation results of simulation 3, scenario 1

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 5.07, -\ 2.60)$
	TCP 2	$0.99 : (-\ 5.78, -\ 3.06)$
	TCP 3	$0.99 : (-\ 5.78, -\ 2.43)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 10.47, -\ 9.05)$
	TCP 2	$0.99 : (-\ 10.48, -\ 9.32)$
	TCP 3	$0.99 : (-\ 10.31, -\ 8.83)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 3.30, -\ 1.81)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 5.45, -\ 4.47)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.03, -\ 0.02)$

Table B.14: Statistical evaluation of the EFCM simulation results of simulation 3, scenario 2

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 13.68, -\ 12.21)$
	TCP 2	$0.99 : (-\ 14.20, -\ 11.95)$
	TCP 3	$0.99 : (-\ 13.66, -\ 11.93)$
$\overline{TPC}$ of a new TCP connection [$1-\alpha$: conf()]	TCP 1	$0.99 : (-\ 9.81, -\ 8.34)$
	TCP 2	$0.99 : (-\ 9.53, -\ 8.38)$
	TCP 3	$0.99 : (-\ 9.50, -\ 8.27)$
$\overline{TPO}$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 8.96, -\ 7.89)$
$\overline{TPC}$ of a concurrent TCP connection [$1-\alpha$: conf()]		$0.99 : (-\ 6.85, -\ 5.96)$
$\overline{I}_f$ of concurrent TCP connections [$1-\alpha$: conf()]		$0.99 : (-\ 0.04, -\ 0.03)$

Table B.15: Statistical evaluation of the EFCM simulation results of simulation 4, scenario 1

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections $[1-\alpha$: conf()]	TCP 1	$0.99 : (-\ \ 4.28, -\ \ 3.35)$
	TCP 2	$0.99 : (-\ \ 5.04, -\ \ 3.71)$
	TCP 3	$0.99 : (-\ \ 4.40, -\ \ 3.23)$
$\overline{TPC}$ of a new TCP connection $[1-\alpha$: conf()]	TCP 1	$0.99 : (-\ \ 7.71, -\ \ 7.22)$
	TCP 2	$0.99 : (-\ \ 8.05, -\ \ 7.30)$
	TCP 3	$0.99 : (-\ \ 7.70, -\ \ 7.13)$
$\overline{TPO}$ of concurrent TCP connections $[1-\alpha$: conf()]		$0.99 : (-\ \ 2.63, -\ \ 2.18)$
$\overline{TPC}$ of a concurrent TCP connection $[1-\alpha$: conf()]		$0.99 : (-\ \ 4.10, -\ \ 3.70)$
$\overline{I}_f$ of concurrent TCP connections $[1-\alpha$: conf()]		$0.99 : (-\ \ 0.04, -\ \ 0.03)$

Table B.16: Statistical evaluation of the EFCM simulation results of simulation 4, scenario 2

		no EFCM ↔ EFCM
$\overline{TPO}$ of new TCP connections $[1-\alpha$: conf()]	TCP 1	$0.99 : (-\ \ 4.77, -\ \ 4.44)$
	TCP 2	$0.99 : (-\ \ 4.70, -\ \ 4.39)$
	TCP 3	$0.99 : (-\ \ 4.70, -\ \ 4.30)$
$\overline{TPC}$ of a new TCP connection $[1-\alpha$: conf()]	TCP 1	$0.99 : (-\ \ 4.19, -\ \ 3.96)$
	TCP 2	$0.99 : (-\ \ 4.11, -\ \ 3.87)$
	TCP 3	$0.99 : (-\ \ 4.16, -\ \ 3.83)$
$\overline{TPO}$ of concurrent TCP connections $[1-\alpha$: conf()]		$0.99 : (-\ \ 2.88, -\ \ 2.71)$
$\overline{TPC}$ of a concurrent TCP connection $[1-\alpha$: conf()]		$0.99 : (-\ \ 2.70, -\ \ 2.54)$
$\overline{I}_f$ of concurrent TCP connections $[1-\alpha$: conf()]		$0.99 : (-\ \ 0.04, -\ \ 0.03)$

Appendix C

Transient Analysis Part II

C.1 Transient Analysis Method

The method used for the transient analysis of the simulation results in Part II is the same used for the transient analysis of Part I. It is already described in Appendix A in detail.

C.2 Transient Analysis Results

As an example, the following Figures C.1 and C.3 show the transient analysis for the throughput of TCP, ETCP, or XCP connections, respectively, traversing the base station of radio cell 6 (cf. Figure 15.1) in a simulation scenario with lower mean network load. Figures C.2 and C.4 show the same l-Δ_l plots in more detail.

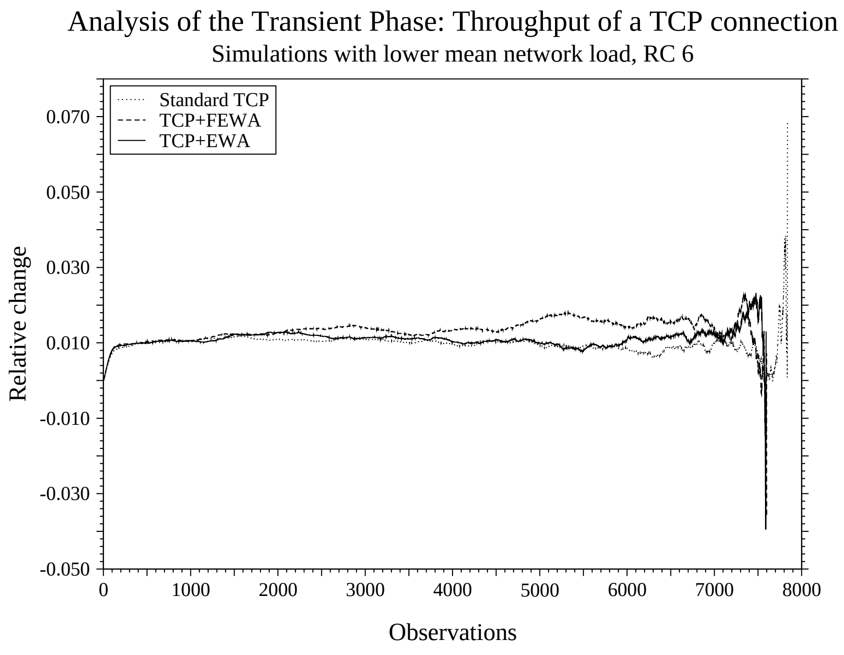


Figure C.1: Transient analysis for the EWA and FEWA simulations of Part II with lower mean network load

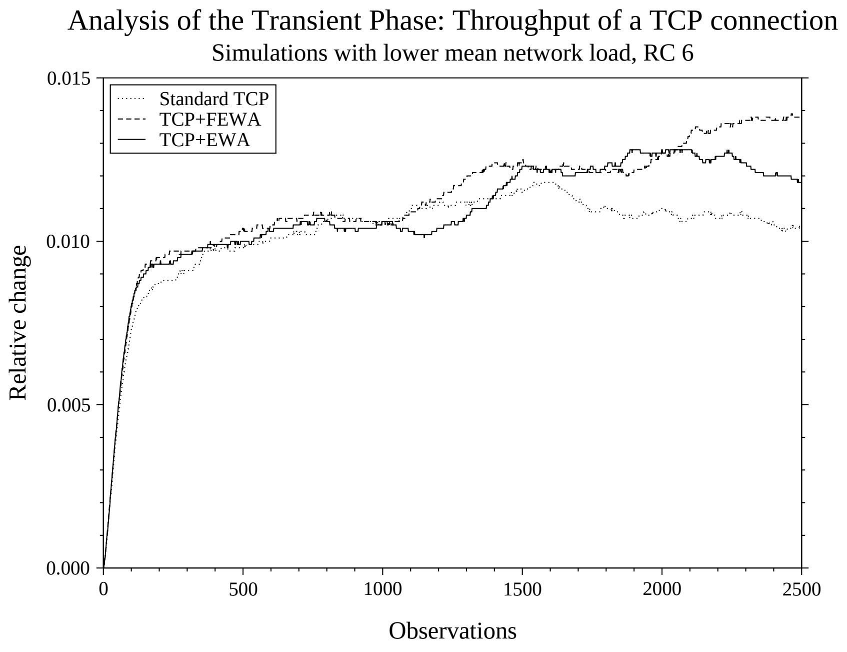


Figure C.2: Zoomed transient analysis for the EWA and FEWA simulations of Part II with lower mean network load

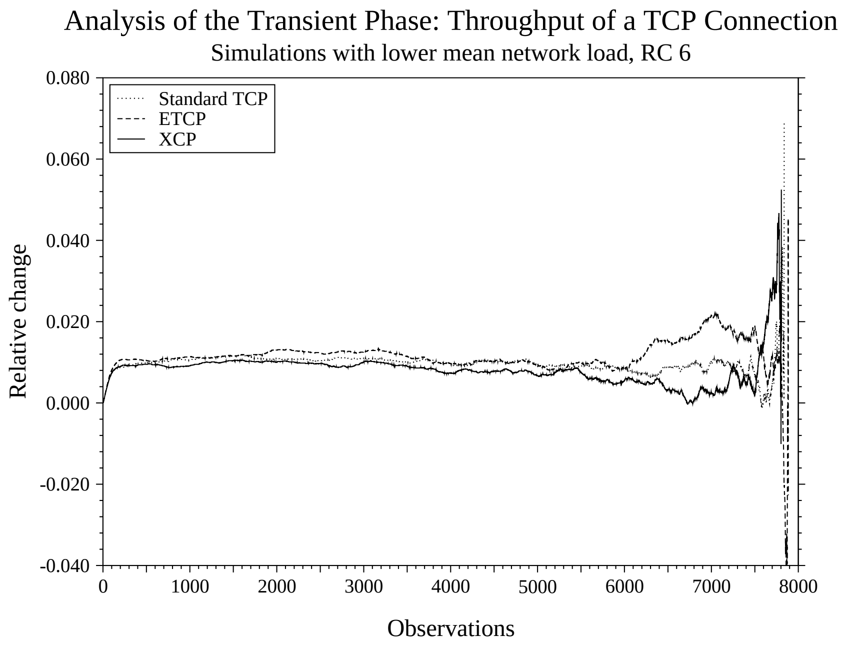


Figure C.3: Transient analysis for the ETCP and XCP simulations of Part II with lower mean network load

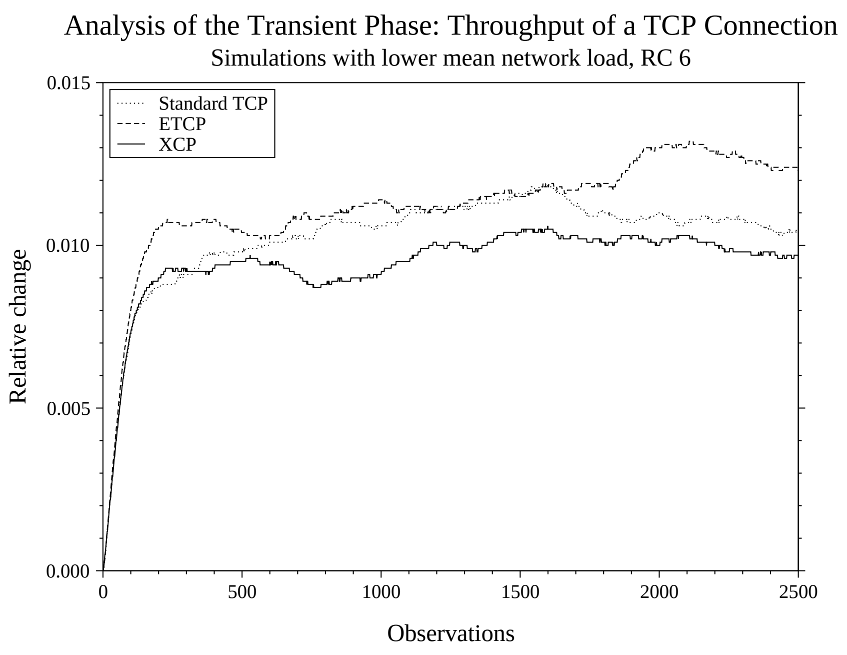


Figure C.4: Zoomed transient analysis for the ETCP and XCP simulations of Part II with lower mean network load

Another example pictured in Figures C.5 and C.7 show the transient analysis for the throughput of TCP, ETCP, or XCP connections, respectively, traversing the base station of radio cell 6 (cf. Figure 15.1) in a simulation scenario with higher mean network load. Figures C.6 and C.8 show the same l-Δ_l plots in more detail.

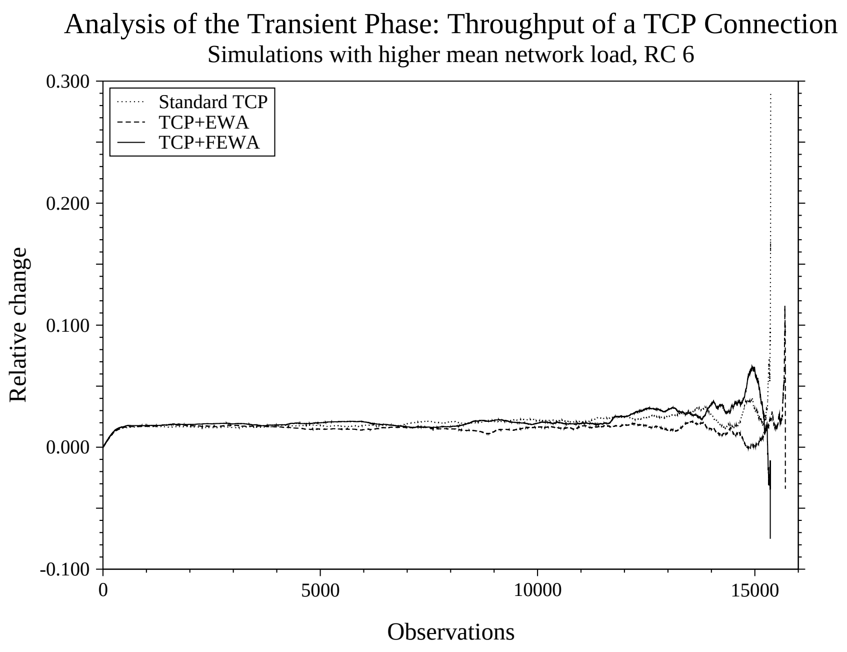


Figure C.5: Transient analysis for the EWA and FEWA simulations of Part II with higher mean network load

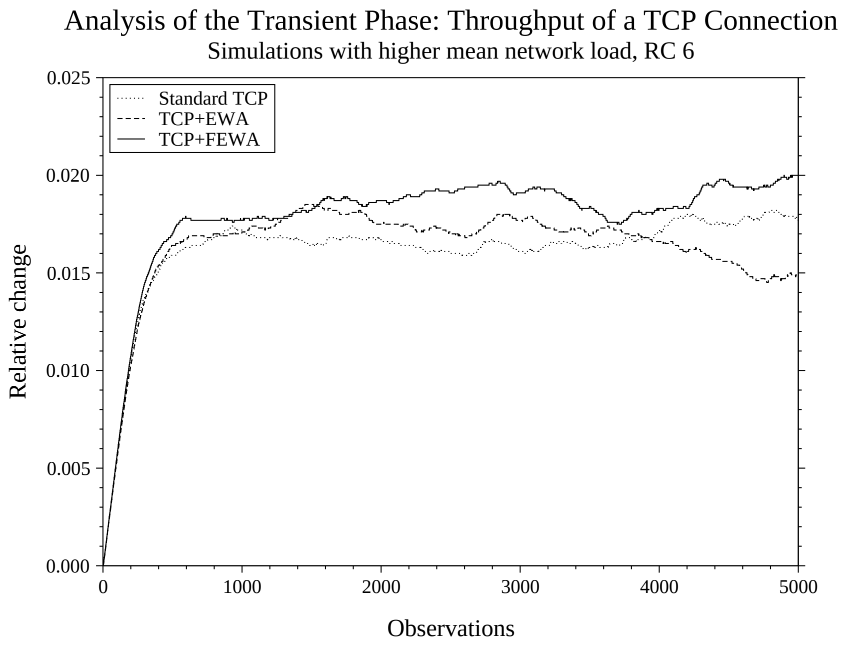


Figure C.6: Zoomed transient analysis for the EWA and FEWA simulations of Part II with higher mean network load

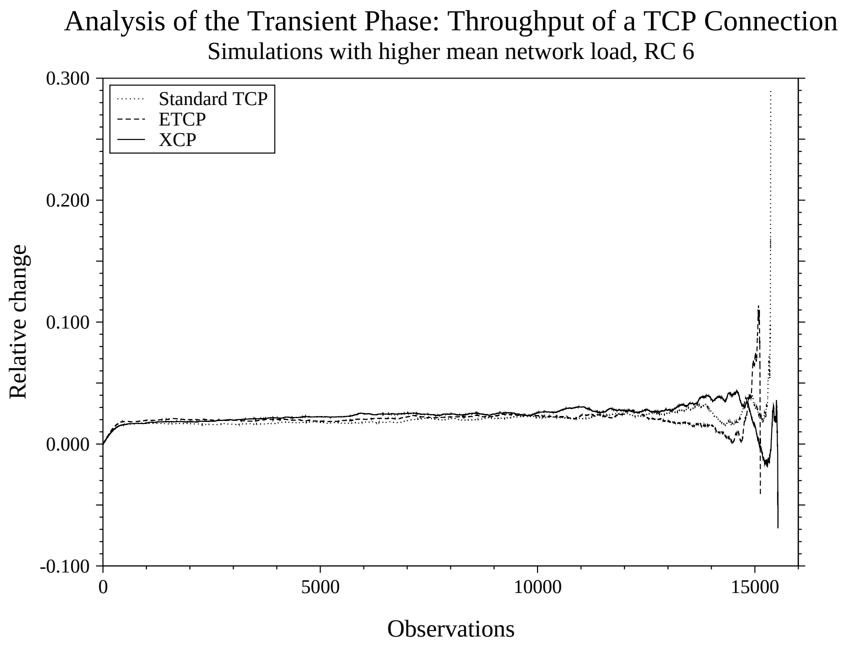


Figure C.7: Transient analysis for the ETCP and XCP simulations of Part II with higher mean network load

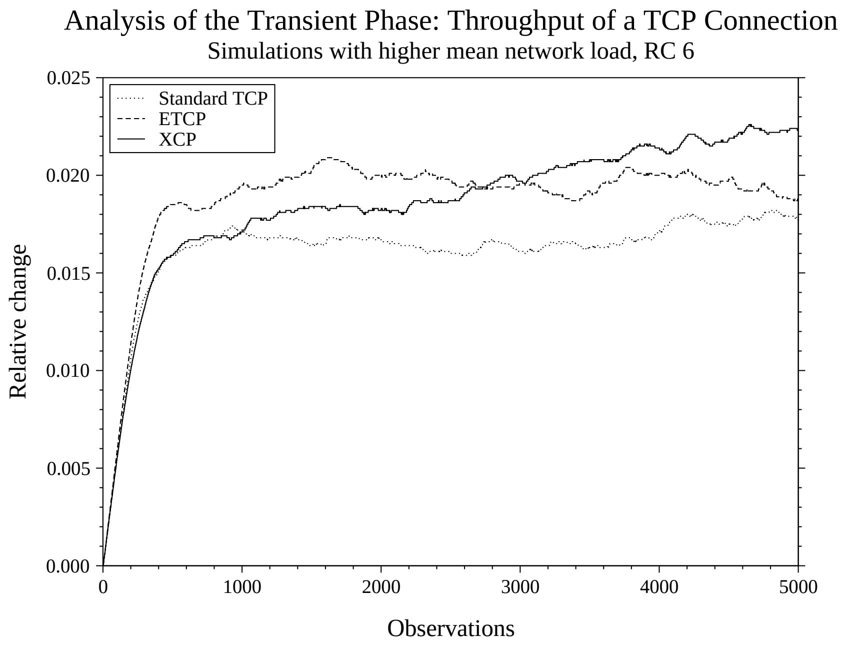


Figure C.8: Zoomed transient analysis for the ETCP and XCP simulations of Part II with higher mean network load

In both examples and in other simulation scenarios of Part II, a transient phase of approximately 10 % of all TCP or XCP throughput observations can be stated. This can be converted to a transient phase of less than 10 % of the whole simulated time of 18000 s. Therefore, a transient phase of 1800 s has been determined. The mean values of the performance metrics are calculated by using all TCP or XCP throughput observations and packet loss events after the transient phase of 1800 s and leave out all TCP or XCP throughput observations and packet loss events during this transient phase.

Appendix D

Statistical Evaluation Part II

D.1 Statistical Evaluation Method

The main ideas of the statistical evaluation method t-test used for this comparison have been already described in Section B.1.

This t-test for unpaired observations of two alternatives is used for the statistical evaluation of the simulation results for both the overall mean throughput ($\overline{TPO}$) and the connection-oriented mean throughput ($\overline{TPC}$) for TCP connections with receivers in each of the (W)LANs and the mean number of packet losses in the base station of each (W)LAN.

The values for this statistical evaluation are produced by twenty independent simulation runs of the considered simulation scenario.

D.2 Statistical Evaluation Results for EWA and FEWA Simulations

In the following Tables D.1 and D.2 the statistical evaluation of the simulation results are shown. For each confidence interval also the confidence level (0.90, 0.95 or 0.99) is depicted. If the simulation results for standard TCP, standard TCP assisted by EWA, or standard TCP assisted by FEWA are not significantly different even for the confidence level 0.90, then the confidence interval for the confidence level 0.90 is stated.

Table D.1: Statistical evaluation of the EWA and FEWA simulation results of scenario 1

		standard TCP ↔ TCP+EWA	standard TCP ↔ TCP+FEWA	TCP+EWA ↔ TCP+FEWA
$\overline{TPO}$ of TCP connections [1 − α : conf()]	(W)LAN 1	$0.90 : (-\ 0.38, +\ 0.19)$	$0.90 : (-\ 0.29, +\ 0.18)$	$0.90 : (-\ 0.23, +\ 0.31)$
	(W)LAN 2	$0.90 : (-\ 0.38, +\ 0.14)$	$0.90 : (-\ 0.18, +\ 0.37)$	$0.90 : (-\ 0.09, +\ 0.52)$
	(W)LAN 3	$0.90 : (-\ 0.06, +\ 0.46)$	$0.90 : (-\ 0.13, +\ 0.44)$	$0.90 : (-\ 0.31, +\ 0.22)$
	(W)LAN 4	$0.90 : (-\ 0.39, +\ 0.23)$	$0.99 : (-\ 1.39, -\ 0.42)$	$0.99 : (-\ 1.25, -\ 0.40)$
	(W)LAN 5	$0.90 : (-\ 0.36, +\ 0.22)$	$0.99 : (-\ 1.53, -\ 0.60)$	$0.99 : (-\ 1.47, -\ 0.52)$
	(W)LAN 6	$0.90 : (-\ 0.57, +\ 0.12)$	$0.99 : (-\ 1.31, -\ 0.35)$	$0.99 : (-\ 1.19, -\ 0.03)$
$\overline{TPC}$ of TCP connections [1 − α : conf()]	(W)LAN 1	$0.90 : (-\ 0.18, +\ 0.06)$	$0.90 : (-\ 0.17, +\ 0.03)$	$0.90 : (-\ 0.13, +\ 0.11)$
	(W)LAN 2	$0.90 : (-\ 0.15, +\ 0.12)$	$0.90 : (-\ 0.11, +\ 0.15)$	$0.90 : (-\ 0.11, +\ 0.18)$
	(W)LAN 3	$0.90 : (-\ 0.08, +\ 0.16)$	$0.90 : (-\ 0.07, +\ 0.18)$	$0.90 : (-\ 0.11, +\ 0.14)$
	(W)LAN 4	$0.90 : (-\ 0.08, +\ 0.22)$	$0.90 : (-\ 0.26, +\ 0.07)$	$0.90 : (-\ 0.32, -\ 0.02)$
	(W)LAN 5	$0.90 : (-\ 0.10, +\ 0.20)$	$0.90 : (-\ 0.27, +\ 0.06)$	$0.90 : (-\ 0.32, +\ 0.02)$
	(W)LAN 6	$0.90 : (-\ 0.26, +\ 0.14)$	$0.90 : (-\ 0.22, +\ 0.17)$	$0.90 : (-\ 0.19, +\ 0.26)$
mean number of packet losses in base station [1 − α : conf()]	(W)LAN 1	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 2	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 3	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 4	$0.99 : (+\ 1329.62, +\ 1923.98)$	$0.99 : (+\ 2296.08, +\ 2866.12)$	$0.99 : (+\ 845.53, +\ 1063.07)$
	(W)LAN 5	$0.99 : (+\ 1406.58, +\ 1964.62)$	$0.99 : (+\ 2329.37, +\ 2867.23)$	$0.99 : (+\ 814.12, +\ 1011.28)$
	(W)LAN 6	$0.99 : (+\ 1384.33, +\ 1955.57)$	$0.99 : (+\ 2283.15, +\ 2839.85)$	$0.99 : (+\ 802.83, +\ 980.27)$

Table D.2: Statistical evaluation of the EWA and FEWA simulation results of scenario 2

		standard TCP $\leftrightarrow$ TCP+EWA	standard TCP $\leftrightarrow$ TCP+FEWA	TCP+EWA $\leftrightarrow$ TCP+FEWA
$\overline{TPO}$ of TCP connections $[1-\alpha$: conf()]	(W)LAN 1	$0.90 : (-0.37, +0.22)$	$0.90 : (-0.52, +0.07)$	$0.90 : (-0.44, +0.14)$
	(W)LAN 2	$0.99 : (-0.54, +0.39)$	$0.90 : (-0.41, +0.03)$	$0.90 : (-0.51, +0.04)$
	(W)LAN 3	$0.90 : (+0.04, +0.56)$	$0.90 : (-0.27, +0.34)$	$0.90 : (-0.56, +0.03)$
	(W)LAN 4	$0.90 : (-0.21, +0.74)$	$0.99 : (-1.74, -0.23)$	$0.99 : (-1.99, -0.51)$
	(W)LAN 5	$0.90 : (-0.33, +0.61)$	$0.99 : (-2.19, -0.58)$	$0.99 : (-2.39, -0.65)$
	(W)LAN 6	$0.90 : (-0.42, +0.53)$	$0.99 : (-2.05, -0.45)$	$0.99 : (-2.03, -0.58)$
$\overline{TPC}$ of TCP connections $[1-\alpha$: conf()]	(W)LAN 1	$0.90 : (-0.13, +0.15)$	$0.90 : (-0.27, +0.04)$	$0.90 : (-0.28, +0.04)$
	(W)LAN 2	$0.90 : (-0.16, +0.11)$	$0.90 : (-0.21, +0.02)$	$0.90 : (-0.21, +0.07)$
	(W)LAN 3	$0.90 : (+0.02, +0.25)$	$0.90 : (-0.14, +0.18)$	$0.90 : (-0.28, +0.04)$
	(W)LAN 4	$0.90 : (-0.05, +0.52)$	$0.90 : (-0.16, +0.39)$	$0.90 : (-0.40, +0.16)$
	(W)LAN 5	$0.90 : (+0.00, +0.57)$	$0.90 : (-0.36, +0.29)$	$0.90 : (-0.64, +0.01)$
	(W)LAN 6	$0.90 : (-0.15, +0.40)$	$0.90 : (-0.37, +0.27)$	$0.90 : (-0.46, +0.11)$
mean number of packet losses in base station $[1-\alpha$: conf()]	(W)LAN 1	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 2	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 3	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 4	$0.99 : (+9022.65, +11386.65)$	$0.99 : (+13670.93, +16010.47)$	$0.99 : (+4394.15, +4877.95)$
	(W)LAN 5	$0.99 : (+9493.13, +11486.17)$	$0.99 : (+14211.41, +16181.69)$	$0.99 : (+4478.07, +4935.73)$
	(W)LAN 6	$0.99 : (+9035.85, +11673.05)$	$0.99 : (+13677.97, +16283.13)$	$0.99 : (+4344.95, +4907.25)$

D.3 Statistical Evaluation Results for ETCP and XCP Simulations

In the following Tables D.3 and D.4 the statistical evaluation of the simulation results are shown. For each confidence interval also the confidence level (0.90, 0.95 or 0.99) is depicted. If the simulation results for standard TCP, ETCP, or XCP are not significantly different even for the confidence level 0.90, then the confidence interval for the confidence level 0.90 is stated.

Table D.3: Statistical evaluation of the ETCP and XCP simulation results of scenario 1

		standard TCP ↔ ETCP	standard TCP ↔ XCP	ETCP ↔ XCP
$\overline{TPO}$ of TCP connections $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.99 : (-\ 7.06, -\ 5.16)$	$0.99 : (-\ 1.75, -\ 1.05)$	$0.99 : (+\ 3.77, +\ 5.64)$
	(W)LAN 2	$0.99 : (-\ 7.46, -\ 5.19)$	$0.99 : (-\ 1.49, -\ 0.68)$	$0.99 : (+\ 4.09, +\ 6.39)$
	(W)LAN 3	$0.99 : (-\ 6.85, -\ 5.07)$	$0.99 : (-\ 1.50, -\ 0.61)$	$0.99 : (+\ 4.01, +\ 5.80)$
	(W)LAN 4	$0.99 : (-\ 6.41, -\ 5.04)$	$0.99 : (+\ 2.50, +\ 3.44)$	$0.99 : (+\ 8.02, +\ 9.37)$
	(W)LAN 5	$0.99 : (-\ 6.44, -\ 5.24)$	$0.99 : (+\ 2.38, +\ 3.32)$	$0.99 : (+\ 8.07, +\ 9.31)$
	(W)LAN 6	$0.99 : (-\ 6.32, -\ 5.25)$	$0.99 : (+\ 2.56, +\ 3.41)$	$0.99 : (+\ 8.24, +\ 9.31)$
$\overline{TPC}$ of TCP connections $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.99 : (-\ 11.23, -\ 10.81)$	$0.99 : (-\ 3.73, -\ 3.43))$	$0.99 : (+\ 7.24, +\ 7.64)$
	(W)LAN 2	$0.99 : (-\ 11.24, -\ 10.79)$	$0.99 : (-\ 3.62, -\ 3.22)$	$0.99 : (+\ 7.37, +\ 7.83)$
	(W)LAN 3	$0.99 : (-\ 11.11, -\ 10.67)$	$0.99 : (-\ 3.62, -\ 3.22)$	$0.99 : (+\ 7.24, +\ 7.69)$
	(W)LAN 4	$0.99 : (-\ 6.52, -\ 5.74)$	$0.99 : (+\ 3.98, +\ 4.46)$	$0.99 : (+\ 9.95, +\ 10.74)$
	(W)LAN 5	$0.99 : (-\ 6.59, -\ 5.83)$	$0.99 : (+\ 4.01, +\ 4.53)$	$0.99 : (+\ 10.10, +\ 10.85)$
	(W)LAN 6	$0.99 : (-\ 6.54, -\ 5.93)$	$0.99 : (+\ 3.98, +\ 4.51)$	$0.99 : (+\ 10.17, +\ 10.78)$
mean number of packet losses in base station $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 2	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 3	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$	$0.90 : (-\ 0.00, +\ 0.00)$
	(W)LAN 4	$0.99 : (+\ 2296.08, +\ 2866.12)$	$0.99 : (+\ 2296.08, +\ 2866.12)$	$0.90 : (+\ 0.00, +\ 0.00)$
	(W)LAN 5	$0.99 : (+\ 2329.37, +\ 2867.23)$	$0.99 : (+\ 2329.37, +\ 2867.23)$	$0.90 : (+\ 0.00, +\ 0.00)$
	(W)LAN 6	$0.99 : (+\ 2283.15, +\ 2839.85)$	$0.99 : (+\ 2283.15, +\ 2839.85)$	$0.90 : (-\ 0.00, +\ 0.00)$

Table D.4: Statistical evaluation of the ETCP and XCP simulation results of scenario 2

		standard TCP $\leftrightarrow$ ETCP	standard TCP $\leftrightarrow$ XCP	ETCP $\leftrightarrow$ XCP
$\overline{TPO}$ of TCP connections $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.90 : (-0.58, +1.10)$	$0.99 : (+0.21, +1.45)$	$0.90 : (-0.30, +1.44)$
	(W)LAN 2	$0.90 : (-1.13, +0.73)$	$0.99 : (+0.34, +1.28)$	$0.90 : (+0.05, +1.96)$
	(W)LAN 3	$0.90 : (-0.68, +1.23)$	$0.99 : (+0.70, +1.76)$	$0.90 : (-0.01, +1.93)$
	(W)LAN 4	$0.99 : (-4.57, -2.83)$	$0.99 : (+1.77, +3.10)$	$0.99 : (+5.35, +6.93)$
	(W)LAN 5	$0.99 : (-4.39, -2.88)$	$0.99 : (+1.57, +2.89)$	$0.99 : (+5.11, +6.62)$
	(W)LAN 6	$0.99 : (-4.93, -3.27)$	$0.99 : (+1.50, +2.91)$	$0.99 : (+5.58, +7.03)$
$\overline{TPC}$ of TCP connections $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.99 : (-9.48 -8.87)$	$0.99 : (-1.11, -0.57)$	$0.99 : (+7.98, +8.68)$
	(W)LAN 2	$0.99 : (-9.56, -9.06)$	$0.99 : (-1.17, -0.59)$	$0.99 : (+8.11, +8.76)$
	(W)LAN 3	$0.99 : (-9.76, -9.08)$	$0.99 : (-1.04, -0.56)$	$0.99 : (+8.27, +8.97)$
	(W)LAN 4	$0.99 : (-5.00, -3.93)$	$0.99 : (+4.00, +4.82)$	$0.99 : (+8.37, +9.37)$
	(W)LAN 5	$0.99 : (-4.85, -3.92)$	$0.99 : (+3.89, +4.81)$	$0.99 : (+8.25, +9.22)$
	(W)LAN 6	$0.99 : (-5.29, -4.21)$	$0.99 : (+3.84, +4.70)$	$0.99 : (+8.53, +9.51)$
mean number of packet losses in base station $[1-\alpha : \text{conf}()]$	(W)LAN 1	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 2	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 3	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$	$0.90 : (-0.00, +0.00)$
	(W)LAN 4	$0.99 : (+13653.46, +15980.84)$	$0.99 : (+13605.46, +15948.44)$	$0.90 : (-46.47, +173.97)$
	(W)LAN 5	$0.99 : (+14211.41, +16181.69)$	$0.99 : (+14139.45, +16123.55)$	$0.90 : (-47.42, +177.52)$
	(W)LAN 6	$0.99 : (+13677.33, +16268.07)$	$0.99 : (+13615.70, +16227.80)$	$0.90 : (-42.87, +160.47)$

Appendix E

Parameters of Fuzzy Explicit Window Adaptation (FEWA)

In this appendix, the linguistic rules of the FEWA algorithm are presented. Furthermore, the chosen parameters of the membership functions $m_{\Delta Q}$ and $m_{\Delta G}$ of the linguistic variables ΔQ and ΔG are stated.

E.1 Linguistic Rules of FEWA

If the queue (length) is empty,
then the utilization factor should be very high. (R1)

If the queue (length) is short and
the rate of change is decreasing fast, (R2)
then the utilization factor should be high.

If the queue (length) is short and
the rate of change is decreasing slowly, (R3)
then the utilization factor should be high.

If the queue (length) is short and
the rate of change is zero, (R4)
then the utilization factor should be high.

If the queue (length) is short and
the rate of change is increasing slowly, (R5)
then the utilization factor should be high.

If the queue (length) is short and
the rate of change is increasing fast, (R6)
then the utilization factor should be medium.

If the queue (length) is moderate and
the rate of change is decreasing fast, (R7)
then the utilization factor should be high.

If the queue (length) is moderate and
the rate of change is decreasing slowly, (R8)
then the utilization factor should be medium.

If the queue (length) is moderate and
the rate of change is zero, (R9)
then the utilization factor should be medium.

If the queue (length) is moderate and
the rate of change is increasing slowly, (R10)
then the utilization factor should be medium.

If the queue (length) is moderate and
the rate of change is increasing fast, (R11)
then the utilization factor should be little.

If the queue (length) is long and
the rate of change is decreasing fast, (R12)
then the utilization factor should be little.

If the queue (length) is long and
the rate of change is decreasing slowly, (R13)
then the utilization factor should be little.

If the queue (length) is long and
the rate of change is zero, (R14)
then the utilization factor should be very little.

If the queue (length) is long and
the rate of change is increasing slowly, (R15)
then the utilization factor should be very little.

If the queue (length) is long and
the rate of change is increasing fast, (R16)
then the utilization factor should be very little.

If the queue (length) is full and
the rate of change is decreasing fast, (R17)
then the utilization factor should be very little.

If the queue (length) is full and
the rate of change is decreasing slowly, (R18)
then the utilization factor should be very little.

If the queue (length) is full and
the rate of change is zero, (R19)
then the utilization factor should be very little.

If the queue (length) is full and
the rate of change is increasing slowly, (R20)
then the utilization factor should be very little.

If the queue (length) is full and
the rate of change is increasing fast, (R21)
then the utilization factor should be very little.

If the queue (length) is congested,
then the utilization factor should be very very little. (R22)

E.2 Parameters of FEWA

Each row k of the following tables E.1, E.2 shows the angle points $(x_{k,1}; y_{k,1}), \dots, (x_{k,4}; y_{k,4})$ of the membership function m_{v_k} of the linguistic value v_k, $1 \leq k \leq n_v$.

Table E.1: Parameters of the linguistic variable $\Delta Q = Q/QT$ (see Figure 13.1)

k	$m_{\Delta Q_k}$
1	$(\ 0.00; 1), (\ 0.00; 1), (\ 0.20; 1), (\ 0.40; 0)$
2	$(\ 0.30; 0), (\ 0.50; 1), (\ 0.80; 1), (\ 0.90; 0)$
3	$(\ 0.80; 0), (\ 0.90; 1), (\ 1.10; 1), (\ 1.20; 0)$
4	$(\ 1.10; 0), (\ 1.20; 1), (\ 1.40; 1), (\ 1.60; 0)$
5	$(\ 1.50; 0), (\ 1.70; 1), (\ 1.90; 1), (\ 2.00; 0)$
6	$(\ 1.90; 0), (\ 2.00; 1), (\ 2.00; 1), (\ 2.00; 1)$

Table E.2: Parameters of the linguistic variable $\Delta G = G/B$ (see Figure 13.2)

k	$m_{\Delta G_k}$
1	$(-1.00; 1), (-1.00; 1), (-0.20; 1), (-0.15; 0)$
2	$(-0.20; 0), (-0.15; 1), (-\ 0.10; 1), (-0.05; 0)$
3	$(-0.10; 0), (-0.05; 1), (\ 0.05; 1), (\ 0.10; 0)$
4	$(\ 0.05; 0), (\ 0.10; 1), (\ 0.15; 1), (\ 0.20; 0)$
5	$(\ 0.15; 0), (\ 0.20; 1), (\ 1.00; 1), (\ 1.00; 1)$

Notice:

- If $(x_{k,1}; y_{k,1})$ is equal to $(x_{k,2}; y_{k,2})$, then $m_{v_k}(x) = y_{k,1}$ for all $x \leq x_{k,1}$.
- If $(x_{k,3}; y_{k,3})$ is equal to $(x_{k,4}; y_{k,4})$, then $m_{v_k}(x) = y_{k,4}$ for all $x \geq x_{k,4}$.

E.3 Selection of α_i's for Different Maximum Queue Lengths

In this section, a rule of thumb for the calculation of the α_i's will be derived that can be used for queues with a maximum queue length different from $B = 99$. One assumption for this rule of thumb is that only B is changed. All other parameters, linguistic variables, and linguistic rules of FEWA are unchanged, at least in relation to the new maximum queue length $B^{'}$.

Let $\alpha = (\alpha_1, \ldots, \alpha_6)$ be the parameter set for the FEWA fuzzy controller of a queue with a maximum queue length B, e.g., $B = 99$. For another queue with a maximum queue length $B^{'}$, e.g., $B^{'} = 999$, the parameter set $\alpha^{'} = (\alpha_1^{'}, \ldots, \alpha_6^{'})$ of this queue is selected that the control surface of the new FEWA fuzzy controller matches the control surface shown in Figure 13.3. Thus,

$$\alpha \cdot \log_2 (B - Q) = \alpha^{'} \cdot \log_2 \left(B^{'} - Q^{'}\right) \tag{E.1}$$

for comparable current queue lengths $Q = f_m \cdot B$ and $Q^{'} = f_m \cdot B^{'}$, $0 \leq f_m \leq 1$, expressed using the maximum queue length or $Q = f_t \cdot QT$ and $Q^{'} = f_t \cdot QT^{'}$, $0 \leq f_t \leq T = B/QT = B^{'}/QT^{'}$, expressed using the target queue length. Then

$$\alpha \cdot \log_2 ((1 - f_m) \cdot B) = \alpha^{'} \cdot \log_2 \left((1 - f_m) \cdot B^{'}\right) \tag{E.2}$$

This is equivalent–at least for queue lengths where only one membership function applies–to

$$\alpha_k \cdot \log_2 ((1 - f_k) \cdot B) = \alpha_k^{'} \cdot \log_2 \left((1 - f_k) \cdot B^{'}\right) \qquad 1 \leq k \leq 6 \tag{E.3}$$

with $x_{k,1} \cdot T^{-1} \leq f_k \leq x_{k,4} \cdot T^{-1}$ (cf. Table E.1). It follows that

$$\alpha_k^{'} = \frac{\log_2((1 - f_k) \cdot B)}{\log_2((1 - f_k) \cdot B')} \cdot \alpha_k = \frac{\log_2(1 - f_k) + \log_2(B)}{\log_2(1 - f_k) + \log_2(B')} \cdot \alpha_k \qquad 1 \leq k \leq 6 \tag{E.4}$$

But this equation has no unique solution for possible values of f_k if B and $B^{'}$ differ. What can be done is to solve this equation by selecting a single value as a representative for every validity interval of a membership function, e.g.,

$$f_k^* = \frac{x_{k,2} + x_{k,3}}{2} \cdot T^{-1} \qquad 1 \leq k \leq 6, \tag{E.5}$$

and calculate an approximation of

$$\alpha_k^{'} \approx \frac{\log_2(1 - f_k^*) + \log_2(B)}{\log_2(1 - f_k^*) + \log_2(B')} \cdot \alpha_k = \theta_k^{'} \cdot \alpha_k \qquad 1 \leq k \leq 6 \tag{E.6}$$

for this membership function by using the single value. This is the rule of thumb.

E.3.1 Example: Persistent and WWW Traffi c Traversing a Single Bottleneck Router

For a maximum queue length $B = 99$ the parameter set $\alpha = (1, 2, 4, 6, 9, 15)$ has been identified as a good choice. Based on this result, the parameter set of a queue with a maximum queue length $B^{'} = 999$ is $\alpha^{'} = (0.66, 1.31, 2.60, 3.87, 5.70, 9.42)$.

The following Figure E.1 shows the histograms of the queue with $B = 999$ of a highly loaded router for the two RCF approaches EWA and FEWA compared to standard TCP. This example simulation

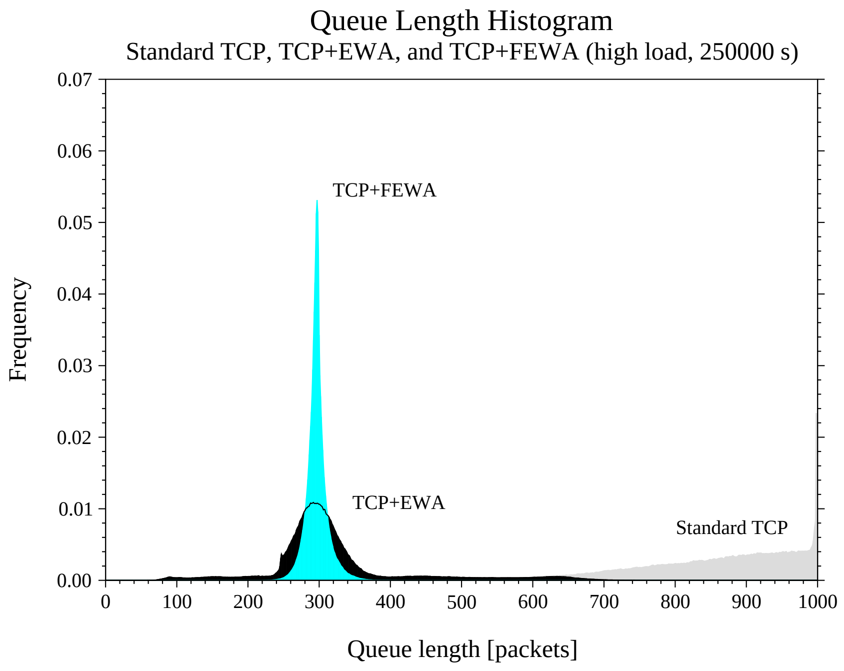


Figure E.1: Queue length process in a highly loaded bottleneck router for standard TCP, TCP+EWA, and TCP+FEWA

scenario considers a traffic mixture of persistent and WWW-based TCP connections which is related to the simulation scenarios considered in [88, 89]. But even in this good-case scenario for EWA, FEWA is able to outperform EWA with a performance gain of more than 25 % considering the overall mean throughput. And compared to standard TCP, FEWA reaches a 590 % larger overall mean throughput.

Appendix F

Variables and Parameters of the Congestion Feedback Approaches

In this chapter, an overview is given about the variables and parameters used in the RCF approaches considered in this dissertation. Some of the variables and parameters of the RCF algorithms are used with indices to further specify a flow or a time interval. These indices are omitted in the following tables.

F.1 EWA

Table F.1: Variables and Parameters of EWA

Variable / Parameter	Description	Unit	(Initial) Value
B	maximum queue length	packets	e.g. 99
i	control interval	s	e.g. 0.010
Q	current queue length	packets	0
$\overline{Q}$	average current queue length	packets	0
α	utilization factor	1	1
w_{up}	AI-parameter	1	1/8
w_{down}	MD-parameter	1	31/32
$\text{threshold}_{\text{low}}$	lower queue threshold	packets	$0.20 \cdot B$
$\text{threshold}_{\text{high}}$	upper queue threshold	packets	$0.60 \cdot B$
MSS	maximum segment size	bytes	e.g. 1460

F.2 FEWA

Table F.2: Variables and Parameters of FEWA

Variable / Parameter	Description	Unit	(Initial) Value
B	maximum queue length	packets	e.g. 99
i	control interval	s	e.g. 0.010
Q	current queue length	packets	0
QT	target queue length	packets	24
$(\Delta)Q$	(fractional) queue length	1	see Table E.1
$(\Delta)G$	(fractional) queue length growth rate	1	see Table E.2
α	utilization factor	1	15
α_k	parameters in the FLC	1	1, 2, 4, 6, 9, 15
MSS	maximum segment size	bytes	e.g. 1460

F.3 XCP

Table F.3: Variables and Parameters of XCP

Variable / Parameter	Description	Unit	(Initial) Value
r	desired sending rate	bytes/s	
cwnd	congestion window	bytes	
s	packet size	bytes	e.g. 1500
rtt	round trip time	s	
d	average RTT of flows	s	
S	spare bandwidth	bytes	
α	weight for $d \cdot S$	1	0.4
Q	persistent queue size	bytes	0
β	weight for Q	1	0.226
ϕ	aggregated congestion feedback	bytes	
h	amount of shuffled bandwidth	bytes	
y	input traffic in d	bytes	
γ	fraction of bandwidth for shuffling	1	0.1
p_i	positive feedback	bytes	
n_i	negative feedback	bytes	
ξ_p	variable for p-computing	bytes/s^2	
ξ_n	variable for n-computing	1/s	

F.4 CSFQ

Table F.4: Variables and Parameters of CSFQ

Variable / Parameter	Description	Unit	(Initial) Value
$\widehat{r}$	estimated arrival rate	bytes/s	
t	arrival time of a packet	s	
l	length of a packet	bytes	
K, K_α, K_c	measurement intervals	s	$\approx 2 \cdot$ max queueing-delay
$\widehat{A}$	estimated aggregated arrival rate	bytes/s	
C	output link speed	bytes/s	e.g. 100 Mbps
$\widehat{F}$	accepted aggregated traffic rate	bytes/s	
$\widehat{\alpha}$	estimated fair share	bytes/s	
P	packet drop probability	1	$0 \leq P \leq 1$
label	(new) packet label	bytes/s	1 kbps...65 Mbps $\pm$6.25 %

F.5 FBA-TCP

Table F.5: Variables and Parameters of FBA-TCP

Variable / Parameter	Description	Unit	(Initial) Value
$\widehat{r}$	estimated arrival rate	bytes/s	
t	arrival time of a packet	s	
l	length of a packet	bytes	
K, K_α, K_c	measurement intervals	s	$\approx 2 \cdot$ max queueing-delay
$\widehat{A}$	estimated aggregated arrival rate	bytes/s	
C	output link speed	bytes/s	e.g. 100 Mbps
$\widehat{F}$	accepted aggregated traffic rate	bytes/s	
$\widehat{\alpha}$	estimated fair share	bytes/s	
P	packet drop probability	1	$0 \leq P \leq 1$
label	(new) packet label	bytes/s	1 kbps...65 Mbps $\pm$6.25 %
RTT	round trip time	s	

F.6 QS-TCP

Table F.6: Variables and Parameters of QS-TCP

Variable / Parameter	Description	Unit	(Initial) Value
QS TTL	TTL in QS request	1	$\in_R [0, 255]$
IR	initial rate	packets/s	≤ 2550
RTT	round trip time	s	
MSS	maximum segment size	bytes	e.g. 1460
CWND	congestion window	bytes	

Bibliography

[1] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley & Sons, 1991.

[2] G. Siegmund. *ATM – Die Technik des Breitband-ISDN*, volume 2. R. v. Decker, 1994.

[3] M. de Pruecker. *Asynchronous Transfer Mode - Die Lösung für Breitband-ISDN*, volume 2. Prentice Hall, 1994.

[4] R. Jain. Congestion control and traffic management in ATM networks: Recent advances and a survey. *Computer Networks and ISDN Systems*, 28(13):1723–1738, 1996.

[5] M. Savorić. Untersuchung von Überlastabwehrverfahren für ABR-Dienste in drahtlosen ATM-Netzen. Master's thesis, University of Frankfurt, September 1998.

[6] The ATM Forum. Traffic management specification version 4.1. AF-TM-0121.000, March 1999.

[7] U. R. Krieger and M. Savorić. Performance evaluation of ABR flow-control protocols in a wireless ATM network. *Wireless Networks*, 9(1):73–84, January 2003.

[8] J. Postel. Internet protocol. RFC 791, September 1981.

[9] J. Postel. Transmission control protocol. RFC 793, September 1981.

[10] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[11] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.

[12] D. Comer. *Internetworking withTCP/IP—Principles, Protocols, and Architectures*, volume 1. Prentice Hall, 2000.

[13] R. Jain. Myths about congestion management in high speed networks. *nternetworking: Research and Experience*, 3:101–113, 1992.

[14] 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, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309, April 1998.

[15] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168, September 2001.

[16] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. RFC 2068, May 1996.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol — HTTP/1.1. RFC 2616, 1999.

[18] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services*. Prentice Hall, 2002.

[19] J. Postel and J. K. Reynolds. File transfer protocol. RFC 959, October 1985.

[20] ITU-T. Packet-based multimedia communications systems. H.323, July 2003.

[21] J. Postel. User datagram protocol. RFC 768, August 1980.

[22] A. S. Tanenbaum. *Computer Networks*, volume 4. Prentice Hall, 2002.

[23] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison Wesley, 4 edition, 1998.

[24] J. Schiller. *Mobilkommunikation*. Addison-Wesley, 2000.

[25] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 2460, December 1998.

[26] H. Wiese. *Das neue Internetprotokoll IPv6*. Hanser, 2002.

[27] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, December 1998.

[28] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. RFC 1519, September 1993.

[29] S. Kent and R. Atkinson. Security architecture for the Internet protocol. RFC 2401, November 1998.

[30] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

[31] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanov. TCP selective acknowledgment options. RFC 2018, October 1996.

[32] V. Paxson and M. Allman. Computing TCP's retransmission timer. RFC 2988, November 2000.

[33] M. Hassan and R. Jain. *High Performance TCP/IP Networking—Concepts, Issues, and Solutions*. Prentice Hall, 2004.

[34] R. Ludwig and K. Sklower. The Eifel retransmission timer. *ACM Computer Communications Review*, 30(3), July 2000.

[35] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[36] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, January 1997.

[37] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. RFC 3390, October 2002.

[38] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM*, pages 314–329, August 1988.

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

[40] S. S. Kunniyur. AntiECN marking: A marking scheme for high bandwidth delay connections. In *Proceedings of IEEE International Conference on Communications (ICC), Alaska*, May 2003.

[41] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. RFC 3135, July 2001.

[42] Comments on the e2e mailing list about the router strategies RED, ECN, dropping packets, etc. http://www.postel.org/pipermail/end2end-interest/2004-April/004083.html.

[43] IP Mobility Support for IPv4. RFC 3344, August 2002.

[44] A. Bakre and B. R. Badrinath. Implementation and performance evaluation of Indirect-TCP. *IEEE Transactions on Computers*, 46(3):260–278, March 1997.

[45] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long thin networks. RFC 2757, January 2000.

[46] M. Schläger, B. Rathke, S. Bodenstein, and A. Wolisz. Advocating a remote socket architecture for Internet access using wireless LANs. *Journal of Mobile Networks and Applications*, 6(1):23–42, 2001.

[47] L. Brakmo, S. O'Malley, and L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *Proceedings of the ACM SIGCOMM*, pages 24–35, August 1994.

[48] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. *Wireless Networks*, 8(5):467–479, 2002.

[49] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, April 2003.

[50] S. Floyd. Highspeed TCP for large congestion windows. RFC 3649, December 2003.

[51] C. Jin, D. X. Wei, and S. H. Low. Fast TCP: Motivation, architecture, algorithms, performance. In *Proceedings of IEEE INFOCOM*, March 2004.

[52] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communications Review*, 27(3):67–82, July 1997.

[53] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[54] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. RFC 3448, January 2003.

[55] M. Allman. A web server's view of the transport layer. *ACM Computer Communication Review*, 30(5):10–20, October 2000.

[56] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in IP traffic. In *Proceedings of the 2nd Internet Measurement Workshop (IMW 2002), Marseille*, pages 253–266, November 2002.

[57] J. Touch. TCP control block interdependence. RFC 2140, 1997.

[58] K. Egevang and P. Francis. The IP network address translator (NAT). RFC 1631, May 1994.

[59] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. *IEEE/ACM Transactions on Networking*, 10(3):381–395, June 2002.

[60] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[61] K. Fall and K. Varadhan. *The ns Manual*. VINT Project, http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf, 2001.

[62] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared passive network performance discovery. In *Proceedings USITS97*, pages 135–146, 1997.

[63] L. Eggert, J. Heidemann, and J. Touch. Effects of ensemble TCP. *ACM SIGCOMM Computer Communication Review*, 30(1):15–29, 2000.

[64] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of ACM SIGCOMM'99*, pages 175–187, 1999.

[65] H. Balakrishnan and S. Seshan. The congestion manager. RFC 3124, 2001.

[66] R. L. Carter and M. E. Crovella. Measuring bottleneck-link speed in packet-switched networks. Technical Report BU-CS-96-006, Computer Science Department, Boston University, March 1996.

[67] S. Keshav. Packet-pair flow control. *IEEE/ACM Transactions on Networking*, February 1995.

[68] H. Balakrishnan, S. Seshan, M. Stemm, and R. H. Katz. Analyzing stability in wide-area network performance. In *Proceedings of ACM SIGMETRICS*, jun 1997.

[69] R. Braden. T/TCP — TCP extentions for transactions. RFC 1644, 1994.

[70] V. Visweswaraiah and J. Heidemann. Improving restart of idle TCP connections. Technical Report 97-661, University of California, 1997.

[71] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz. TCP behavior of a busy Internet server: Analysis and improvements. In *Proceedings of IEEE INFOCOM'98*, pages 252–262, 1998.

[72] U. Schöning. *Algorithmen—kurz gefaßt*. Spektrum Akademischer Verlag, 1997.

[73] K. Mehlhorn. *Datenstrukturen und Effiziente Algorithmen: Sortieren und Suchen*. Springer, 1986.

[74] M. Savorić. The TCP control block interdependence in fixed networks — some performance results. In *Proceedings QOFIS 2001*, LNCS 2156, pages 261–272, 2001.

[75] M. Savorić, H. Karl, and A. Wolisz. The TCP control block interdependence in fixed networks — new performance results. *Computer Communications*, 26(4):366–375, February 2003.

[76] M. Savorić and H. Karl. Performance evaluation of an improved common congestion controller for TCP connections — new simulation results. Technical report, TKN, http://www-tkn.ee.tu-berlin.de/publications/papers/efcm_tr_3.pdf, 2003.

[77] M. Savorić, H. Karl, and A. Wolisz. Selected properties of a joint congestion controller for TCP connections. In *Proc. of ITC18*, pages 861–870, September 2003.

[78] M. Savorić and H. Karl. Validation of some ensemble flow congestion management control algorithms. Technical report, TKN, http://www-tkn.ee.tu-berlin.de/publications/papers/efcm_av_tr.pdf, 2003.

[79] K. Bosch. *Elementare Einführung in die Wahrscheinlichkeitsrechnung*. Vieweg, 6 edition, 1995.

[80] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. In *Proceedings of ACM SIGCOMM*, August 2000.

[81] A. Reyes-Lecuona, E. Gonzáles-Parada, E. Casilari, and J. Casasola. A page-oriented WWW traffic model for wireless system simulations. In *Proceedings ITC 16*, pages 1271–1280, 1999.

[82] J. Belopilski. Implementation and performance evaluation of a TCP common congestion controller in the Linux environment. Master's thesis, TKN, TU Berlin, November 2003.

[83] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolski. An extension to the selective acknowledgment (SACK) option for TCP. RFC 2883, July 2000.

[84] D. Borman, R. Braden, and V. Jacobson. TCP extensions for high performance. RFC 1323, May 1992.

[85] R. Ludwig and R. H. Katz. The Eifel algorithm: Making TCP more robust against spurious retransmissions. *ACM Computer Communications Review*, 30(1):30–36, January 2000.

[86] P. Sarolathi and A. Kuznetsov. Congestion control in linux TCP. In *Proceedings of USENIX 2002 Annual Technical Conference*, pages 49–62, June 2002.

[87] J. Hoe. Startup dynamics of TCP's congestion control and avoidance schemes. Master's thesis, Massachusetts Institute of Technology, 1995.

[88] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Explicit window adaptation: A method to enhance TCP performance. In *In Proceedings of IEEE INFOCOM'98*, pages 242–251, 1998.

[89] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Explicit window adaptation: A method to enhance TCP performance. *IEEE/ACM Transactions on Networking*, 10(3):338–350, June 2002.

[90] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM'02*, pages 89–102, August 2002.

[91] D. Katabi. *Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks*. PhD thesis, Massachussetts Institute of Technology, March 2003.

[92] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM'98*, pages 118–130, 1998.

[93] R. Kapoor, C. Casetti, and M. Gerla. Core-stateless fair bandwidth allocation for TCP flows. In *Proceedings of IEEE ICC 2001*, 2001.

[94] A. K. Jain and S. Floyd. Quick-start for TCP and IP. http://www.ietf.org/internet-drafts/draft-amit-quick-start-02.txt, work in progress, October 2002.

[95] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM'00*, pages 43–56, 2000.

[96] M. Savorić and U. R. Krieger. Adaptation of the fuzzy explicit rate marking to ABR flow-control in a wireless ATM network. In *Proc. of MMB99*, pages 25–30, September 1999.

[97] M. Savorić, U. R. Krieger, and A. Wolisz. The impact of handover protocols on the performance of ABR flow-control algorithms in a wireless ATM network. *European Transactions on Telecommunications (ETT)*, 11:419–430, August 2000. Special Issue on Service Quality Control in Multimedia Wireless Networks.

[98] M. Savorić and U. R. Krieger. The performance of a fuzzy flow-control scheme for WWW data transfer in a wireless ATM network. In *Proc. of ITC Specialist Seminar on Mobile Systems and Mobility*, pages 145–156, March 2000.

[99] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller—part I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, March 1990.

[100] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller—part II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):419–435, March 1990.

[101] A. Pitsillides, Y. A. Şekercioğlu, and G. Ramamurthy. Effective control of traffic flow in ATM networks using fuzzy explicit rate marking (FERM). *IEEE Journal on Selected Areas in Communications*, 15(2):209–225, 1997.

[102] G. Schäfer. *Netzsicherheit*. dpunkt-Verlag, 2003.

[103] M. Savorić. Description and comparison of distributed congestion management approaches in IP-based networks. project report EF-TUB-D6, March 2003.

[104] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of IEEE INFOCOM'00*, pages 1537–1545, 2000.

[105] M. Savorić. Distributed congestion management approaches in IP-based networks — first performance results for WCDMA links. project report EF-TUB-D8, December 2003.

[106] P. Tittmann. *Einführung in die Kombinatorik*. Spektrum Akademischer Verlag, 2000.

[107] M. R. Spiegel and J. Liu. *Mathematical Handbook of Formulas and Tables*. Schaum's, 2 edition, 1999.

[108] B. Welch. *Practical Programming in Tcl and Tk*. Prentice Hall, 3 edition, 2000.

[109] P. Raines. *Tcl/Tk—kurz & gut*. O'Reilly, 1998.

[110] K. Günther. *LaTeX gepackt*. mitp-Verlag, 2002.