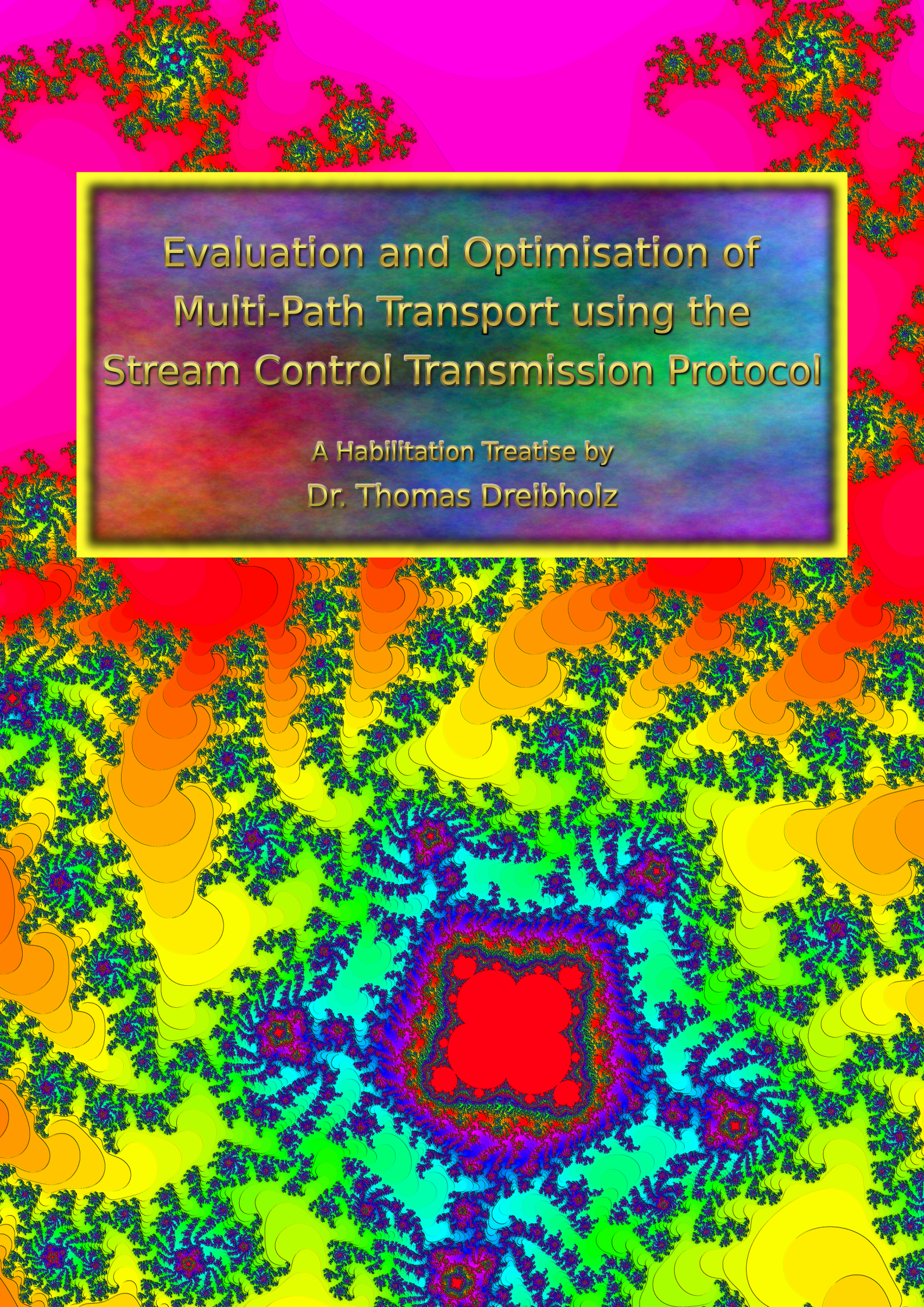

Evaluation and Optimisation of
Multi-Path Transport using the
Stream Control Transmission Protocol
A Habilitation Treatise by
Dr. Thomas Dreibholz

Evaluation and Optimisation of Multi-Path Transport using the Stream Control Transmission Protocol

H A B I L I T A T I O N T R E A T I S E

in Computer Science

Submitted to the
Faculty of Economics and Business Administration
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen

by
Dr. Thomas Dreibholz
born on September 29, 1976 in
Bergneustadt, Nordrhein-Westfalen, Germany

President of the University of Duisburg-Essen:
Prof. Dr. Ulrich Radtke

Dean of the Faculty of Economics and Business Administration:
Prof. Dr. Michael Goedicke

Reviewers:

1. Prof. Dr.-Ing. Erwin P. Rathgeb
2. Prof. Dr.-Ing. Ralf Steinmetz
3. Prof. Dr. Paul Müller

Submitted on: September 8, 2011
Date of Publication: March 13, 2012

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig ohne fremde Hilfe verfaßt und nur die angegebene Literatur und Hilfsmittel verwendet zu haben.

Thomas Dreibholz
March 13, 2012

Abstract

The Stream Control Transmission Protocol (SCTP) as defined in RFC 4960 is an advanced Transport Layer protocol that provides support for multi-homing. That is, SCTP endpoints may simultaneously use multiple Network Layer addresses, which allows to connect the endpoints to multiple networks for redundancy purposes. However, for the transfer of user data, only one of the possible paths is currently used at a time. All other paths remain as backup and are only used for retransmissions.

Clearly, the existence of multiple paths has led to the idea of applying load sharing among the paths. An extension to SCTP – denoted as Concurrent Multipath Transfer (CMT) – realises this load sharing functionality. While this approach works well for similar paths, i.e. paths having similar characteristics regarding bandwidths, bit error rates and delays, the use of dissimilar paths does not work that neatly.

In this thesis, the issues of dissimilar paths for CMT-based load sharing will be demonstrated first. The reasons for these issues will be identified and solutions proposed. These solutions will be evaluated in simulations, as well as partially also in a real-world Internet testbed setup, in order to show their effectiveness. In particular, it will be shown that a combination of multiple mechanisms is necessary to make CMT work as expected under a wide range of network and system parameters.

Furthermore, the fairness of CMT-based transport – in concurrency to classic non-CMT flows – will be analysed. The usage of plain CMT leads to an overly aggressive bandwidth occupation on so-called shared bottlenecks. As a countermeasure, the idea of Resource Pooling will be utilised. For this purpose, two new and one adapted congestion control approach – all based on the Resource Pooling principle – will be introduced and examined in similar as well as dissimilar path setups, in order to show how to fairly deploy CMT transport in the Internet.

The results of this work have also been contributed to the ongoing IETF standardisation process of SCTP and its extensions.

Keywords:
Stream Control Transmission Protocol (SCTP), Multi-Path Transport, Dissimilar Paths, Fairness, Evaluation, Optimisation

Zusammenfassung

Das Stream Control Transmission Protocol (SCTP), welches im RFC 4960 spezifiziert wurde, ist ein fortgeschrittenes Transport-Layer-Protokoll mit Unterstützung für Multi-Homing. Dies bedeutet, dass aus Redundanzgründen SCTP-Endpunkte an mehrere Netzwerke gleichzeitig angeschlossen sein können. Allerdings wird für die Übertragung von Benutzerdaten immer nur ein einziger der möglichen Pfade verwendet. Alle anderen Pfade verbleiben als Ersatz und werden nur für Wiederholungen (engl. retransmissions) verwendet.

Das Vorhandensein von mehreren Pfaden legt natürlich nahe, Lastverteilung (engl. load sharing) auf den Pfaden einzusetzen. Eine Erweiterung von SCTP – welche als Concurrent Multipath Transfer (CMT) bezeichnet wird – realisiert eben diese Funktionalität. Dieser Ansatz funktioniert gut für gleichartige (engl. similar) Pfade, das heißt Pfade mit ähnlichen Charakteristika wie Bandbreiten, Verzögerungen und Bitfehlerraten. Auf ungleichartigen (engl. dissimilar) Pfaden jedoch zeigen sich erhebliche Performanzprobleme.

In dieser Habilitationsschrift werden zunächst die Schwierigkeiten von CMT-basierter Lastverteilung auf ungleichartigen Pfaden gezeigt. Dabei werden die Ursachen der Probleme aufgezeigt sowie Lösungen vorgeschlagen. Diese Lösungen werden sowohl durch Simulationen als teilweise auch in einem realistischen Internet-Testbettaufbau bewertet, um ihre Leistungsfähigkeit zu verdeutlichen. Im Besonderen wird dabei gezeigt, dass eine Kombination von mehreren Mechanismen notwendig ist, um CMT mit der erwarteten Leistung innerhalb eines großen Bereiches von Netzwerk- und Systemparametern zu betreiben.

Des Weiteren wird die Fairness von CMT-basiertem Transport – im Wettbewerb mit klassischen, Nicht-CMT-Datenströmen, analysiert. Die Verwendung von reinem CMT führt zu einer übermäßig aggressiven Bandbreitenbelegung auf sogenannten gemeinsamen Engpässen (engl. shared bottlenecks). Als Gegenmaßnahme wird die Idee des Resource Poolings aufgegriffen. Zu diesem Zweck werden zwei neue sowie ein angepasster Ansatz zur Überlastkontrolle – alle basierend auf dem Resource-Pooling-Prinzip – eingeführt und sowohl in Szenarien mit gleichartigen als auch ungleichartigen Pfaden untersucht, um zu zeigen wie CMT fair im Internet eingesetzt werden kann.

Die Ergebnisse dieser Arbeit sind zudem in den laufenden IETF-Standardisierungsprozess von SCTP und seinen Erweiterungen eingeflossen.

Schlagwörter:
Stream Control Transmission Protocol (SCTP), Multi-Path-Transport, Ungleichartige Pfade, Fairness, Evaluierung, Optimierung

Acknowledgements

This thesis is the result of my work as assistant professor in the Computer Networking Technology Group of the Institute for Experimental Mathematics at the University of Duisburg-Essen. At this point, I would like to express my acknowledgement to everybody who has supported me during my research on Reliable Server Pooling (RSerPool) as well as on the Stream Control Transmission Protocol (SCTP) during the last decade.

In particular, I would like to thank my thesis advisor, Erwin Paul Rathgeb, for his support of my SCTP research and this thesis, as well as the reviewers Paul Müller and Ralf Steinmetz for having agreed to review this large document within the tight time schedule. Also, I would like to thank the members of my habilitation commission, and in particular the commission chairman Bruno Müller-Clostermann, for the timely processing of my habilitation.

Furthermore, I would like to express my special thanks to my colleagues Hakim Adhari and Martin Becke for their great cooperation with respect to discussions on buffer splitting and congestion control variants as well as configuring and debugging the SCTP testbed environment. I would like to furthermore thank my former colleague Jobin Pulinthanath for his help with building up the testbed environment and negotiating the contract details with the ADSL Internet service provider. For his help with the acquisition of the testbed hardware, I would also like to thank our computer systems technician Nihad Cosić.

Clearly, I would furthermore like to express my special thanks to my colleagues Irene Rüngeler, Robin Seggelmann and Michael Tüxen from the Münster University of Applied Sciences in Burgsteinfurt for their input during the discussions about buffer splitting, their help with setting up and debugging the testbed environment as well as debugging the SCTP simulation model. Also, I would like to thank Randall R. Stewart for the initial discussions about receive buffer splitting.

Also, I would like to thank the Deutsche Forschungsgemeinschaft (DFG) for sponsoring the project on SCTP this thesis has been performed within.

Last but not least, I would like to thank my father Ernst Günter Dreibholz and my mother Annelore Dreibholz for the years of encouragement and support.

http://tdrwww.iem.uni-due.de/dreibholz/sctp/

Contents

Glossary

3GPP	3rd Generation Partnership Project
ACK	Acknowledgement
ADSL	Asymmetric Digital Subscriber Line
AF	Assured Forwarding
AIMD	Additive Increase, Multiplicative Decrease
API	Application Programming Interface
APP	Applications Area (IETF)
ARP	Address Resolution Protocol
ASAP	Aggregate Server Access Protocol
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
ATU – C	ADSL Transceiver Unit – Central Office
ATU – R	ADSL Transceiver Unit – Remote
BDP	Bandwidth-Delay Product
BGP	Border Gateway Protocol
BIPM	Bureau International des Poids et Mesures
BMBF	Bundesministerium für Bildung und Forschung
BRAS	Broadband Remote Access Server
BSD	Berkeley Software Distribution
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CMT	Concurrent Multipath Transfer
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRC – 32/4	Cyclic Redundancy Check, 32 bits, variant 4
CRC – 32C	Cyclic Redundancy Check, 32 bits, Castagnoli
CSP	Component Status Protocol
CumAck	Cumulative Acknowledgement
CVE	Common Vulnerabilities and Exposures
CVS	Concurrent Versions System
cwnd	Congestion Window
DCCP	Datagram Congestion Control Protocol
DFG	Deutsche Forschungsgemeinschaft
DFN	Deutsches Forschungsnetz
DiffServ	Differentiated Services
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
ECMP	Equal-Cost Multi-Path

ECN	Explicit Congestion Notification
EF	Expedited Forwarding
ENRP	Endpoint haNdlespace Redundancy Protocol
FEC	Forward Error Correction
FIFO	First In First Out
FTP	File Transfer Protocol
G – Lab	German Lab
GapAck	Gap Acknowledgement
GEN	General Area (IETF)
GNU	GNU is Not Unix
GPL	GNU General Public License
GPLv3	GNU General Public License, version 3
HIP	Host Identity Protocol
HTTP	HyperText Transfer Protocol
I – D	Internet Draft
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
IAOC	Internet Administrative Oversight Committee
ICI	Interface Control Information
ICMP	Internet Control Message Protocol
ICMPv4	Internet Control Message Protocol, version 4
ICMPv6	Internet Control Message Protocol, version 6
ID	Identifier
IDU	Interface Data Unit
IEC	International Electrotechnical Commission
IEEE	Institute for Electrical and Electronics Engineers
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IMT – 2000	International Mobile Telecommunications 2000
INT	Internet Area (IETF)
IntServ	Integrated Services
IP	Internet Protocol
IPFIX	IP Flow Information Export
IPPM	IP Performance Metrics
IPR	Intellectual Property Rights
IPv4	Internet Protocol, version 4
IPv6	Internet Protocol, version 6
IRTF	Internet Research Task Force
ISBN	International Standard Book Number
ISDN	Integrated Services Digital Network
ISO	International Standards Organisation
ISOC	Internet Society
ISP	Internet Service Provider
ISSN	International Standard Serial Number
ITU	International Telecommunications Union
ITU – D	ITU Telecommunication Development Sector
ITU – R	ITU Radiocommunication Sector

ITU – T	ITU Telecommunication Standardization Sector
KVM	Keyboard/Video/Mouse
LAN	Local Area Network
LGPL	GNU Lesser General Public License
MD5	Message Digest Algorithm No. 5
MPI	Message Passing Interface
MPTCP	Multi-Path TCP
MTU	Maximum Transmission Unit
NAK	Negative Acknowledgement
NED	Network Definition Language (OMNeT++)
NetPerfMeter	Network Performance Meter
NPMP – CONTROL	NETPERFMETER Control Protocol
NPMP – DATA	NETPERFMETER Data Protocol
NR – SACK	Non-Renegable Selective Acknowledgement
NS – 2	Network Simulator 2
NTP	Network Time Protocol
OMNeT++	Objective Modular Network Testbed in C++
OPS	Operations and Management Area (IETF)
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OSPFv2	Open Shortest Path First, version 2
OSPFv3	Open Shortest Path First, version 3
PC	Personal Computer
PCAP	Packet Capture
PCI	Protocol Control Information
PCU	Power Control Unit
PDF	Portable Document Format
PDU	Protocol Data Unit
PE	Pool Element
PHB	Per-Hop Behaviour
PNG	Portable Network Graphics
POTS	Plain Old Telephone System
PPID	Payload Protocol Identifier
PPP	Point-to-Point Protocol
PPPoE	Point-to-Point Protocol over Ethernet
PR	Pool Registrar
PR – H	Home Pool Registrar
PU	Pool User
QDisc	Queuing Discipline
QoS	Quality of Service
RAI	Real-time Applications and Infrastructure Area (IETF)
RED	Random Early Detection
RFC	Request for Comments
RIP	Routing Information Protocol
RIPng	Routing Information Protocol, next generation
RIPv2	Routing Information Protocol, version 2
RP	Resource Pooling

RSerPool	Reliable Server Pooling
RTG	Routing Area (IETF)
RTO	Retransmission Timeout
RTT	Round-Trip Time
RTTVAR	Round-Trip Time Variance
RTX	Retransmission
SACK	Selective Acknowledgement
SAP	Service Access Point
SCTP	Stream Control Transmission Protocol
SDU	Service Data Unit
SEC	Security Area (IETF)
SHA – 1	Secure Hash Algorithm No. 1
SI	Système International d'Unités
SIGTRAN	Signalling Transport Working Group (IETF)
SimProcTC	Simulation Processing Tool-Chain (software package)
SRTT	Smoothed Round-Trip Time
SS	Scripting Service
SS7	Signalling System No. 7
SSH	Secure Shell
SSN	Stream Sequence Number
SSP	Scripting Service Protocol
ssthresh	Slow-Start Threshold
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TSN	Transport Sequence Number
TSV	Transport Area (IETF)
TSVWG	Transport Services Working Group (IETF)
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications Systems
UTF – 8	Universal Character Set Transformation Format – 8-bit
VoIP	Voice over IP
W3C	World Wide Web Consortium
WAN	Wide Area Network
WG	Working Group (IETF)
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

This chapter describes the motivation of this thesis, defines its goals and finally shortly introduces its structure.

1.1 Motivation

Since the dawn of the Internet age, the TCP/IP-based protocols have been designed with one fundamental assumption: an endpoint is connected to a single network and addressed by only one Network Layer address. However, the rising popularity of the Internet has also led to the deployment of resilience-critical applications – e.g. e-commerce, e-health and emergency services – which must eliminate single points of failure. Solutions on the Application Layer and Session Layer – like Reliable Server Pooling described by [DR09, LOTD08] – provide endpoint redundancy, i.e. they take care of handling host failures. However, a Transport Layer solution could provide a very quick resolution of network failures, which is a likely failure scenario (consider e.g. a cable cut due to roadworks or mistakenly unplugged by the charlady). Therefore, a new Transport Layer protocol called *Stream Control Transmission Protocol* (SCTP) had been developed and standardized in [Ste07]. SCTP is a general-purpose, datagram-oriented and reliable Transport Layer protocol which supports multiple addresses per endpoint, i.e. particularly addresses on different interfaces being connected to disjoint networks. Furthermore, the application may configure ordered or unordered delivery for each message independently, i.e. specify whether the remote peer has to ensure the message sequence by reordering out-of-sequence messages in its reception buffer, before forwarding them to the peer application. While SCTP has been initially motivated by the requirements of telephone signalling over IP networks, as described in [JRT02], it is a fully-featured Transport Layer protocol which can replace TCP – as suggested by [DR08c] – for applications benefiting from the advanced features of SCTP.

In the context of SCTP, a *Path* denotes the unidirectional data flow from one endpoint to a given remote Network Layer address of the peer endpoint. An example is shown in Figure 1.1: Endpoint A has three paths (addresses B_1, B_2 and B_3) to its peer Endpoint B, and Endpoint B has three paths (addresses A_1, A_2 and A_3) to Endpoint A. Note, that a break of the radio access link would remove address B_3 from Endpoint B, i.e. the number of paths from Endpoint A to Endpoint B would decrease – but the three paths from Endpoint B to Endpoint A would remain.

Although SCTP supports multiple paths, only a single, designated path is actually used for user data transmission. The other paths remain for backup and are only used for retransmissions. Obviously, the existence of multiple paths leads to the desire of applying *Load Sharing* – as suggested by [DWPW07] – to utilise *all* paths simultaneously.

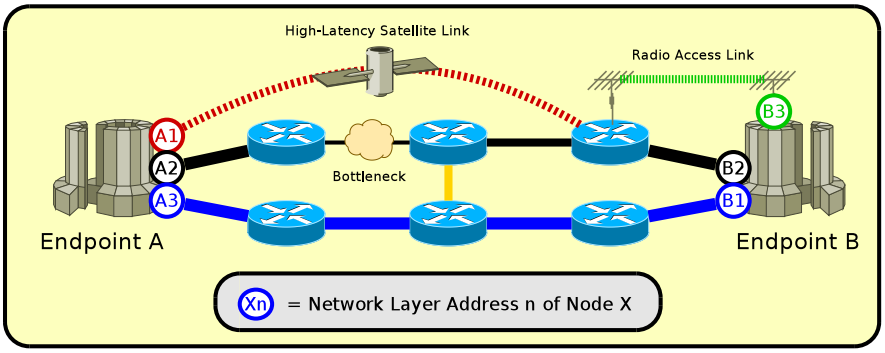


Figure 1.1: Multi-Homed Nodes in a Communications Network

1.2 Scope and Related Work

Load sharing for SCTP has initially been proposed by [ASL03]. However, as discussed by [JR06, Subsubsection 5.1] in detail, this approach adds unnecessary protocol overhead. Ideas for improvements have been evaluated in [Jun05, Chapter 6]; a more advanced approach has been realised by the *Concurrent Multipath Transfer* (CMT) extension described in [IAS06]. At a first glance, SCTP with CMT – denoted as CMT-SCTP – looks quite straightforward and simple. However, load sharing leads to a set of non-trivial issues, which will be described in the following.

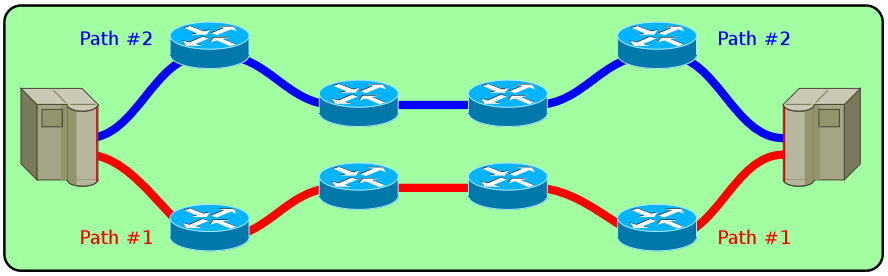
Although CMT-SCTP directly refers to the SCTP protocol, the challenges of load sharing are generic and apply to other multipath transfer approaches as well. Particularly, they are also a topic for *Multi-Path TCP* (MPTCP), which denotes an experimental multipath transfer extension for the TCP protocol that is introduced by [FRH+11]. Therefore, the research on challenges and solutions for efficient multipath-capable transport has become a very actively discussed topic.

1.2.1 Efficient Handling of Dissimilar Paths

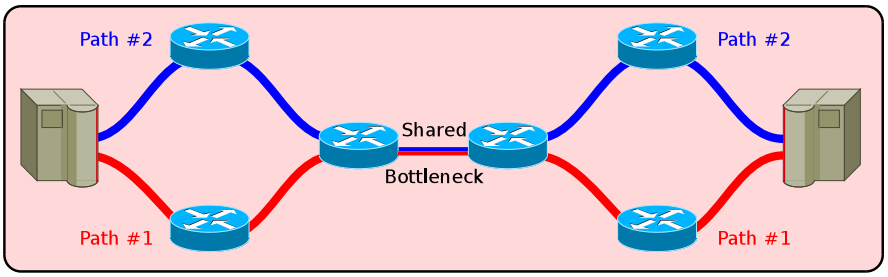
[IAS06] has only examined CMT-SCTP in similar path setups, i.e. for using paths having nearly the same bandwidths, delays and loss rates. However, such similar paths are unlikely in networks like the Internet. Particularly, multi-homed systems – like the example illustrated in Figure 1.1 – usually apply different access technologies (like DSL and UMTS) for redundancy reasons. These technological differences clearly lead to highly dissimilar paths. Initial testbed probing using such dissimilar paths has revealed certain scenarios with a CMT-SCTP throughput even lower than standard, non-CMT SCTP throughput.

1.2.2 Fairness on Shared Bottlenecks

CMT-SCTP as defined by [IAS06] also simply assumes that all paths are disjoint, i.e. they do not share links or routers, as illustrated in Subfigure 1.2(a). While this constraint may be fulfilled by carefully designed networks, it cannot be assured for arbitrary networks like the Internet. Even worse, non-disjoint paths are actually quite likely in current Internet setups, e.g. when tunnelling IPv6 over IPv4. Since the congestion control of CMT-SCTP handles each path independently, a bottleneck shared by two paths – as illustrated in Subfigure 1.2(b) – causes a fairness issue: each of the two paths occupies as much bandwidth as a single non-CMT flow (e.g. a TCP or standard SCTP connection)



(a) Two Disjoint Paths



(b) Two Paths Sharing a Bottleneck Link

Figure 1.2: Multipath Transfer over Disjoint Paths and Shared Bottleneck

over the same bottleneck. That is, CMT-SCTP occupies twice the bandwidth, which is clearly unfair to concurrent non-CMT flows over the same bottleneck. An approach for handling this problem is the idea of *Resource Pooling* by [WHB08], which denotes "to make a collection of resources behave like a single pooled resource". However, a concrete adaptation to CMT-SCTP – and therefore a performance analysis – did not exist.

1.3 Goals

The overall goal of this thesis is – of course – to find solutions for the described challenges of multipath transfer. This work has been performed within in the research project "Systematic Evaluation and Further Development of the Transport Protocol SCTP"[1] (German title: "Systematische Untersuchung und Weiterentwicklung des Transportprotokolls SCTP") funded by the Deutsche Forschungsgemeinschaft[2] (DFG), which is a cooperation between

- the Computer Networking Technology Group[3] at the Institute for Experimental Mathematics of the University of Duisburg-Essen and
- the Department of Electrical Engineering and Computer Science[4] at the Münster University of Applied Sciences.

This thesis can be subdivided into three core parts:

[1] See http://gepris.dfg.de/gepris/OCTOPUS/?module=gepris&task=showDetail&context=projekt&id=62543436.
[2] Deutsche Forschungsgemeinschaft: http://www.dfg.de/.
[3] See http://tdrwww.iem.uni-due.de/.
[4] See https://www.fh-muenster.de/fb2/.

1. Development of a simulation environment and a real testbed as tools for the research work.
2. Investigation and enhancement of the load sharing performance over dissimilar paths.
3. Analysis and improvement of the load sharing fairness.

Besides the research part, it is also intended to bring the developed ideas from research to application – by contributing to the ongoing IETF standardisation process for the SCTP protocol and its extensions. As long-term goal, the work on SCTP is intended to foster the deployment of SCTP, as described further in [DRS+11, DR08c].

1.4 Organisation

This thesis is organised as follows: Chapter 2 provides an introduction to the basic terminology and protocol mechanisms. Furthermore, it gives a brief overview of the protocols being relevant in the context of this thesis. The following Chapter 3 provides an introduction to SCTP and its extensions. Chapter 4 introduces multipath transfer, with a strong focus on the CMT-SCTP extension.

The CMT-SCTP simulation model as well as the simulation processing tool-chain, which have been developed as part of this thesis, are explained in Chapter 5; the real network testbed environment including the developed measurement tool-chain is introduced in Chapter 6.

The analysis of CMT-SCTP-based multipath transfer performance on dissimilar paths can be found in Chapter 7; the examination and improvement of fairness issues is documented in Chapter 8. Finally, a summary of the results and an outlook to future work is provided in Chapter 9.

For the contents of this thesis, colours have been used to enhance the comprehensibility of the figures. It is therefore recommended – but not mandatory – to apply colour printing when reproducing this document. Names of software packages are emphasised by small caps style typesetting (e.g. NETPERFMETER); the names of commands use typewriter style typesetting (e.g. `sysctl`). When terminology is defined, the first occurrence of a newly introduced term is emphasised by italics style typesetting and capitalised words (e.g. *Buffer Splitting*). Also, this first occurrence will be the main reference point for the index. The electronic version of this document makes intensive use of hyper-references. External references are shown in blue colour, references within this document are represented by dark blue colour.

Chapter 2

Basics

This chapter formally defines some basic networking terminology first. After that, important concepts of reference models, services and data communications are introduced. Furthermore, the basic protocol mechanisms for data communications are explained. The chapter closes with an overview of protocol standardisation with a focus on Internet protocols.

2.1 Formal Terminology Definitions

First of all, it is necessary to formally define some basic terminology. A formal definition is necessary here, since terms like networks and paths have different notions when applied in the context of different viewpoints, e.g. from a routing-centric view or a transport-centric view. In order to define an unequivocal terminology – for the discussion of multipath transport in general and this thesis in particular – the methods of discrete mathematics (see [Aig99, DP88]) are applied.

2.1.1 Network

Clearly, the first term to be defined is a network.

Definition 2.1.1. A *network* $\Gamma = (L, N, C, c)$ is defined as:

- L – a finite locator set,
- $N \subseteq \mathfrak{P}(L)$ – a node set,
- $C \subseteq L \times L$ – a connectivity set and
- $c : L \times L \to \mathbb{N}_0 \cup \{\infty\}$ – a cost function.

The following two conditions apply:

1. “Uniqueness of Locators”:

$$\forall n_1, n_2 \in N : [n_1 \cap n_2 \neq \emptyset] \Rightarrow [n_1 = n_2].$$

2. “Forwarding”:

$$\forall n \in N \, \forall \lambda_1, \lambda_2 \in n : [(\lambda_1, \lambda_2) \in C] \Rightarrow [c(\lambda_1, \lambda_2) = 0].$$

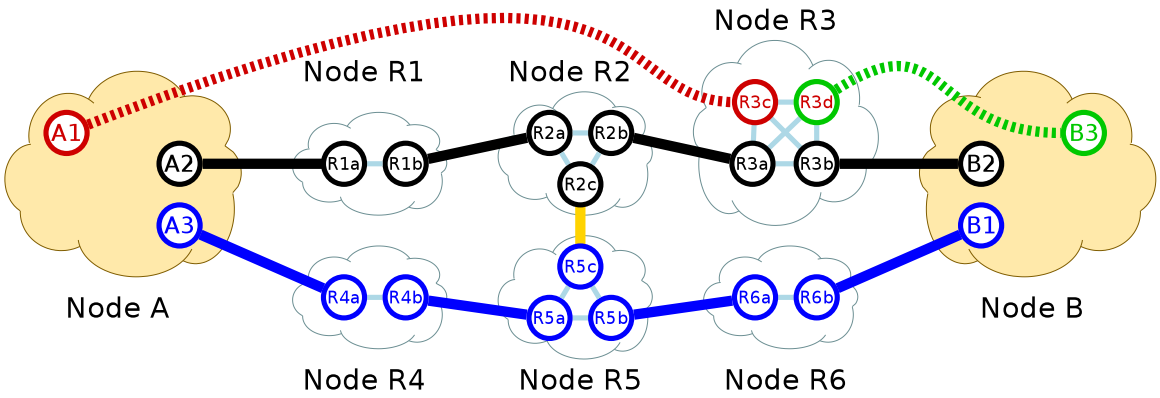


Figure 2.1: An Example Network

The *Locator Set* L defines a finite set of network-unique *Locators*. In a real network, a locator $l \in L$ could e.g. be a Network Interface Card (NIC) or IP address. Then, a *Node* n (e.g. a host in the network) is simply defined by a subset of the locator set ($n \subseteq L$; e.g. the interfaces or IP addresses of the host). The "Uniqueness of Locators" condition ensures that node locators are unique within the network (i.e. two distinct nodes must not use the same locator).

Connectivity among the nodes is described by the *Connectivity Set* $C \subseteq L \times L$. An element $\omega \in C$ is denoted as *Channel*. The cost function c defines its cost; the "Forwarding" condition ensures that connectivity among locators of the same node is complimentary. Note, that a channel between two nodes is directed, i.e. it may just be unidirectional. Also, one physical *Link*, i.e. the actual implementation of a physical data transfer (e.g. an Ethernet), may consist of multiple channels.

For convenience, terminology for two distinct types of nodes is defined:

Definition 2.1.2. Let $\Gamma = (L, N, C, c)$ be a network.

- A node $n \in N$ is denoted as *Router* $\Leftrightarrow \exists \lambda_1 \in n \; \exists \lambda_2 \in n : [\lambda_1 \neq \lambda_2 \wedge (\lambda_1, \lambda_2) \in C]$.
- A node $n \in N$ is denoted as *Endpoint* $\Leftrightarrow n$ is not a router.

That is, a router provides connectivity among at least two of its locators. An example of a network $\Gamma = (L, N, C, c_D)$ is presented in Figure 2.1. It models a network according to the motivation example presented in Figure 1.1. The colours of the nodes and channels have been used accordingly, in order to make the abstraction step from the "real-world setup" visible. L and N are defined as follows:

$$
\begin{aligned}
L &= \{A1, A2, A3, \\
&\quad\; B1, B2, B3, \\
&\quad\; R1a, R1b, R2a, R2b, R2c, R3a, R3b, R3c, R3d, \\
&\quad\; R4a, R4b, R5a, R5b, R5c, R6a, R6b, \\
&\quad\; \Theta 0, \Theta 1\}, \\
N_{\text{Endpoints}} &= \left\{ \underbrace{\{A1, A2, A3\}}_{\text{Node A}}, \underbrace{\{B1, B2, B3\}}_{\text{Node B}} \right\},
\end{aligned}
$$

$$
\begin{aligned}
N_{\text{Routers}} &= \Big\{ \underbrace{\{R1a, R1b\}}_{\text{Node R1}}, \underbrace{\{R2a, R2b, R2c\}}_{\text{Node R2}}, \underbrace{\{R3a, R3b, R3c, R3d\}}_{\text{Node R3}}, \\
&\qquad \underbrace{\{R4a, R4b\}}_{\text{Node R4}}, \underbrace{\{R5a, R5b, R5c\}}_{\text{Node R5}}, \underbrace{\{R6a, R6b\}}_{\text{Node R6}} \Big\}, \\
N &= N_{\text{Endpoints}} \cup N_{\text{Routers}}.
\end{aligned}
$$

Note, that the locator set contains the locators $\Theta 0$ and $\Theta 1$, which are not part of any node (like e.g. spare NICs or unused IP addresses). The connectivity set C of Γ is defined as follows:

$$
\begin{aligned}
C = \{&(A1, R3c), (R3c, A1), (R3d, B3), (B3, R3d), \\
&(A2, R1a), (R1a, A2), (R1b, R2a), (R2a, R1b), \\
&(R2b, R3a), (R3a, R2b), (R3b, B2), (B2, R3b), \\
&(R2c, R5c), (R5c, R2c), \\
&(A3, R4a), (R4a, A3), (R4b, R5a), (R5a, R4b), \\
&(R5b, R6a), (R6a, R5b), (R6b, B1), (B1, R6b), \} \cup \\
&\underbrace{\bigcup_{r \in N_{\text{Routers}}} \bigcup_{\lambda^- \in r} \bigcup_{\lambda^+ \in r} \{(\lambda^-, \lambda^+)\}}_{\text{Forwarding by Routers}} \cup \underbrace{\bigcup_{n \in N} \bigcup_{\lambda \in n} \{(\lambda, \lambda)\}}_{\text{Node-Internal Connectivity}} .
\end{aligned}
$$

The simple cost function "Directly Connected" $c_D : L \times L \to \mathbb{N}_0 \cup \{\infty\}$ is defined as:

$$
(\lambda^-, \lambda^+) \mapsto \begin{cases} 0 & (\exists n \in N : [\lambda^- \in n \wedge \lambda^+ \in n)] \\ 1 & ((\lambda^-, \lambda^+) \in C) \\ \infty & (\text{else}) \end{cases} .
$$

That is, using the cost function c_D, locators of the same node can be reached for free (i.e. cost is zero) and locators of a neighbour can be reached at cost 1. The cost to reach all non-neighbour locators is unknown (i.e. ∞).

For conveniently describing the mapping of a locator to its node (if there is any), it is useful to define a mapping function.

Definition 2.1.3. The "Locator to Node Mapping Function" $\hat{n}$ is defined as:

$$
\begin{aligned}
\hat{n} &: L \to N \\
l &\mapsto \{n \in N | l \in n\} .
\end{aligned}
$$

That is, $\hat{n}$ maps a locator $l \in L$ to $\emptyset$ if it is not part of a node – or to its *only* node if it is part of a node. This uniqueness is a direct implication of the "Uniqueness of Locators" condition. In the example above, $\hat{n}$ maps as follows:

$$
\begin{aligned}
\hat{n}(R3c) &= \{\underbrace{\{R3a, R3b, R3c, R3d\}}_{\text{Node R3}}\}, \\
\hat{n}(\Theta 0) &= \emptyset.
\end{aligned}
$$

2.1.2 Adjacency

For a more convenient description of "node neighbourhood", the definition of four terms is useful:

Definition 2.1.4. Let $\Gamma = (L, N, C, c)$ be a network.

- Locator λ_1 is *Half-Adjacent* to locator λ_2 in $\Gamma \Leftrightarrow (\lambda_1, \lambda_2) \in C$.
- Locator λ_1 is *Adjacent* to locator λ_2 in $\Gamma \Leftrightarrow$

$$\underbrace{[\lambda_1 \text{ is half-adjacent to } \lambda_2]}_{\text{Forwards Direction}} \wedge \underbrace{[\lambda_2 \text{ is half-adjacent to } \lambda_1]}_{\text{Backwards Direction}}.$$

- Node n_1 is *Half-Adjacent* to node n_2 in $\Gamma \Leftrightarrow$

$$\exists \lambda_1 \in n_1 \, \exists \lambda_2 \in n_2 : [\lambda_1 \text{ is half-adjacent to } \lambda_2].$$

  The definition of half-adjacency is particularly useful in wireless networks, e.g. node n_1 may reach node n_2, but the transmission power of node n_2 may be insufficient to reach node n_1.

- Node n_1 is *Adjacent* to node n_2 in $\Gamma \Leftrightarrow$

$$\exists \lambda_1, \overline{\lambda}_1 \in n_1 \, \exists \lambda_2, \overline{\lambda}_2 \in n_2 : \big[\underbrace{[\lambda_1 \text{ is half-adjacent to } \lambda_2]}_{\text{Forwards Direction}} \wedge \underbrace{[\overline{\lambda}_2 \text{ is half-adjacent to } \overline{\lambda}_1]}_{\text{Backwards Direction}} \big].$$

  Note, that the "backwards direction" may use different locators.

2.1.3 Trail

By using the half-adjacency, it is now possible to specify directed trails throughout a network:

Definition 2.1.5. Let $\Gamma = (L, N, C, c)$ be a network. The *Set of Trails* $\overline{T}_\Gamma(n_1, n_2)$ from node n_1 to node n_2 in Γ is defined as:

$$\begin{aligned}
\overline{T}_\Gamma(n_1, n_2) \quad := \quad \Bigg\{ & (\lambda_1, \ldots, \lambda_k) \in L^k \Big| 1 \leq k \leq |L| \wedge \underbrace{\lambda_1 \in n_1}_{\text{Source}} \wedge \underbrace{\lambda_k \in n_2}_{\text{Destination}} \wedge \\
& \underbrace{\forall i : [1 \leq i \leq k-1] \Rightarrow [(\lambda_i, \lambda_{i+1}) \in C]}_{\text{Next Locator in Sequence is Half-Adjacent}} \wedge \\
& \underbrace{\forall i, j : [1 \leq i < j \leq k] \Rightarrow [\lambda_i \neq \lambda_j]}_{\text{Sequence of Locators is Loop-Free}} \wedge \\
& \underbrace{\forall i, m, j : \big[[1 \leq i < m < j \leq k] \wedge [\hat{n}(\lambda_i) = \hat{n}(\lambda_j)]\big] \Rightarrow \big[\hat{n}(\lambda_m) = \hat{n}(\lambda_i)\big]}_{\text{Sequence of Nodes is Loop-Free}} \Bigg\}.
\end{aligned}$$

An element $\tau \in \overline{T}_\Gamma(n_1, n_2)$ is denoted as *Trail* from node n_1 to node n_2 in Γ.

That is, a trail τ from node n_1 to node n_2 is a sequence of half-adjacent locators. The first entry of τ is a locator in n_1 ("source"), the last entry of τ is a locator in n_2 ("destination"). The last part of the conjunction ensures that the trail is loop-free, i.e.

1. each locator may appear at most once and
2. for two locators λ_i and λ_j mapping to the same node (i.e. $\hat{n}(\lambda_i) = \hat{n}(\lambda_j)$), all locators λ_m between (i.e. $i < m < j$) must also map to this node (i.e. $\hat{n}(\lambda_i) = \hat{n}(\lambda_m) = \hat{n}(\lambda_j)$).

In the example depicted in Figure 2.1, τ_1 and τ_2 are trails from Node A to Node B:

$$
\begin{aligned}
\tau_1 &= (A1, R3c, R3b, B2) \\
\tau_2 &= (A2, R1a, R1b, R2a, R2b, R3a, R3d, B3) \\
\tau_3 &= (A1, \underbrace{R3c}_{\text{okay}}, R3a, R3b, \underbrace{R3c}_{\text{}}, R3d, B3) \\
\tau_4 &= (\underbrace{A1}_{\text{Node } A}, R3c, R2b, R2a, R1a, \underbrace{A2, A3}_{\text{Node } A}, R4a, R4b, R5a, R5b, R6a, R6b, B1)
\end{aligned}
$$

On the other hand, τ_3 is not a valid trail – it violates the condition of loop-freeness by passing locator $R3c$ twice. τ_4 passes node A twice: between locators $A1$ and $A2$ – which both belong to Node A – are locators of Node $R3$, Node $R2$ and Node $R1$.

Having a set of trails, it is highly useful to have a cost metric in order to identify e.g. the least-cost trail. Using the cost function c of the network Γ, the cost function $\hat{c}$ for its trails can be defined straightforwardly:

Definition 2.1.6. Let $\Gamma = (L, N, C, c)$ be a network. The *Trail Cost Function* $\hat{c}$ is defined as:

$$
\begin{aligned}
\hat{c} \quad &: \quad \cup_{1 \le k \le |L|} L^k \to \mathbb{N}_0 \cup \{\infty\} \\
(\lambda_1, \ldots, \lambda_k) \quad &\mapsto \quad \sum_{1 \le i \le k-1} c\big((\lambda_i, \lambda_{i+1})\big)
\end{aligned}
$$

In the example above, which uses the "Directly Connected" cost function c_D defined in Subsection 2.1.1, the resulting costs are $\hat{c}(\tau_1) = 2$ and $\hat{c}(\tau_2) = 4$. By using c_D, the trail cost function $\hat{c}$ simply leads to the "Hop Count" metric as defined by [Mal98].

2.1.4 Disjointness of Trails

Comparing two trails τ_1 and τ_2, the existence of a common locator is a crucial property:

Definition 2.1.7. Let $\tau_1 = (\lambda_1^1, \ldots, \lambda_r^1)$ and $\tau_2 = (\lambda_1^2, \ldots, \lambda_s^2)$ be trails in $\overline{T}_\Gamma(n_1, n_2)$. τ_1 and τ_2 are denoted as *Disjoint*

$$
\Leftrightarrow \quad \underbrace{\left[\lambda_1^1 \neq \lambda_1^2\right]}_{\text{Different Source Locators}} \quad \wedge \quad \underbrace{\left[\lambda_r^1 \neq \lambda_s^2\right]}_{\text{Different Destination Locators}} \quad \wedge
$$

$$
\left[\left[\underbrace{\{\hat{n}(\lambda_1^1), \ldots, \hat{n}(\lambda_r^1)\}}_{\text{Nodes of Trail } \tau_1} \cap \underbrace{\{\hat{n}(\lambda_1^2), \ldots, \hat{n}(\lambda_s^2)\}}_{\text{Nodes of Trail } \tau_2} \right] - \left[\underbrace{\{n_1, n_2\}}_{\text{Source and Destination Nodes}} \right] = \emptyset \right].
$$

That is, disjoint trails have different source locators (i.e. $\lambda_1^1 \neq \lambda_1^2$), different destination locators (i.e. $\lambda_r^1 \neq \lambda_s^2$) as well as no common node except for source node n_1 and destination node n_2. In the example depicted in Figure 2.1, there are two disjoint trails between Node A and Node B:

$$\begin{aligned}\tau_1 &= \{ \underbrace{A2}_{\text{Node A}}, \underbrace{R1a, R1b}_{\text{Node R1}}, \underbrace{R2a, R2b}_{\text{Node R2}}, \underbrace{R3a, R3b}_{\text{Node R3}}, \underbrace{B2}_{\text{Node B}} \},\\ \tau_2 &= \{ \underbrace{A3}_{\text{Node A}}, \underbrace{R4a, R4b}_{\text{Node R4}}, \underbrace{R5a, R5b}_{\text{Node R5}}, \underbrace{R6a, R6b}_{\text{Node R6}}, \underbrace{B1}_{\text{Node B}} \}.\end{aligned}$$

On the other hand, τ_1 and the trail

$$\tau_3 = \{ \underbrace{A1}_{\text{Node A}}, \underbrace{R3c, R3d}_{\text{Node R3}}, \underbrace{B3}_{\text{Node B}} \}$$

are non-disjoint, since τ_1 and τ_3 both use Node $R3$.

2.1.5 Path

Definition 2.1.8. Let $\Gamma = (L, N, C, c)$ be a network. The *Set of Paths* $\overline{P}_\Gamma(n_1, n_2)$ from node n_1 to node n_2 in Γ is defined as:

$$\begin{aligned}\overline{P}_\Gamma(n_1, n_2) \;:=\; \big\{ d \in n_2 \;\big|\; &\exists s \in n_1 \; \exists k \in \{0, \ldots, |L|\} \; \exists \lambda_1, \ldots, \lambda_k \in L : \\ &(s, \lambda_1, \ldots, \lambda_k, d) \in \overline{T}_\Gamma(n_1, n_2) \big\}.\end{aligned}$$

An element of $\overline{P}_\Gamma(n_1, n_2)$ is denoted as *Path* from node n_1 to node n_2 in Γ.

That is, a path from node n_1 to a node n_2 is defined as a locator of node n_2, which is reachable by a trail in Γ by using *any* source locator in n_1. The distinction between trail and path decouples the routing (i.e. trails – "how to reach the other node") from node locators (i.e. paths – "the other node is reachable"). In the example depicted in Figure 2.1, there are three paths from Node A to Node B, as well as three paths in the reverse direction:

$$\begin{aligned}\overline{P}_\Gamma(\underbrace{\{A1, A2, A3\}}_{\text{Node A}}, \underbrace{\{B1, B2, B3\}}_{\text{Node B}}) &= \{B1, B2, B3\},\\ \overline{P}_\Gamma(\underbrace{\{B1, B2, B3\}}_{\text{Node B}}, \underbrace{\{A1, A2, A3\}}_{\text{Node A}}) &= \{A1, A2, A3\}.\end{aligned}$$

Note, that e.g. removing the adjacency between $R3d$ and $B3$ from Γ – creating a network Γ^- – would only remove $B3$ from the paths to Node B. The paths in the reverse direction would remain as shown above:

$$\begin{aligned}\overline{P}_{\Gamma^-}(\underbrace{\{A1, A2, A3\}}_{\text{Node A}}, \underbrace{\{B1, B2, B3\}}_{\text{Node B}}) &= \{B1, B2\},\\ \overline{P}_{\Gamma^-}(\underbrace{\{B1, B2, B3\}}_{\text{Node B}}, \underbrace{\{A1, A2, A3\}}_{\text{Node A}}) &= \{A1, A2, A3\}.\end{aligned}$$

2.2 Data Communications

A formal definition for the transfer of data units (e.g. bytes or packets) between endpoints is neglected here, since the concept should be clear and undisputed for the reader. Instead, the following sections describe the basic mechanisms of data transfer in a less formal but more implementation-centric way.

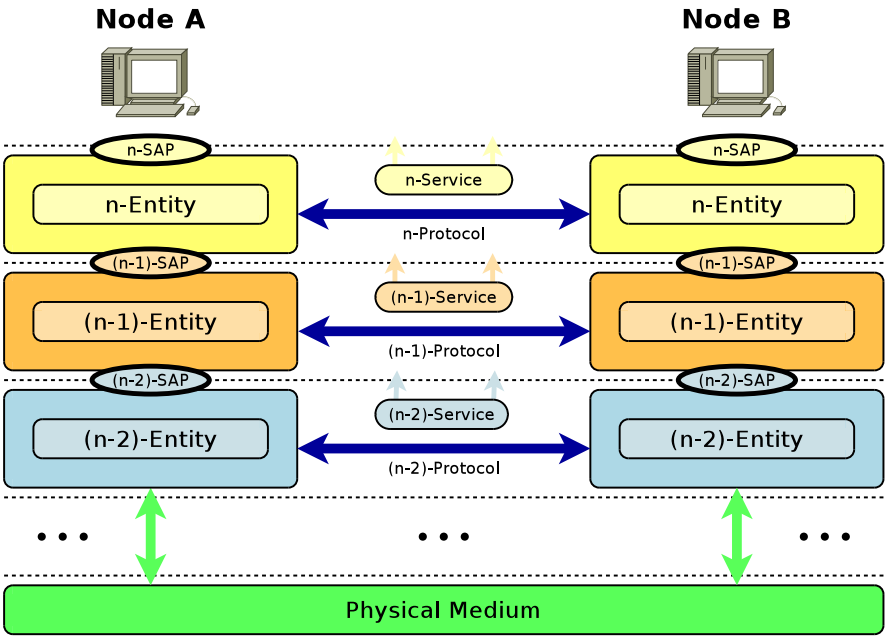


Figure 2.2: The Layered Protocol Stack

2.2.1 Protocols

A data communication between two entities is described by a so-called *Protocol*. A protocol is defined according to [MP93] as a "formal description of message formats and the rules two computers must follow to exchange those messages. Protocols can describe low-level details of machine-to-machine interfaces (e.g. the order in which bits and bytes are sent across a wire) or high-level exchanges between allocation programs (e.g. the way in which two programs transfer a file across the Internet)."

In order to reduce the complexity of a communication system, the communication necessary to perform a sophisticated task – like transferring a file from a node A to a node B – is not realised by a single protocol. Instead, as introduced in [ITU94, Chapter 5], a hierarchy of protocols is used. This so-called *Protocol Stack* is illustrated in Figure 2.2. It is organised in so-called *Layers*. The base of the hierarchy is the so-called *Physical Medium*, which provides an arbitrary way to transfer data units between the lowest layers of both endpoints (e.g. voltage levels on a copper wire). Each layer n uses a well-defined so-called $(n-1)$-*Service* of layer $(n-1)$ and provides its own n-service to layer $(n+1)$. The $(n-1)$-service consists of a set of $(n-1)$-*Service Primitives*, which define operations provided to the layer n. That is, the service primitives define *what* a layer can perform for the next layer, but not *how* it is actually realised. The so-called n-*Entity*, i.e. the actual realisation of layer n on the local endpoint, communicates with the corresponding n-entity on the other endpoint – which is denoted as *Peer* – using the n-protocol.

Here, the layered protocol stack shows its advantages:

Adaptability It is easy to replace complete layers, i.e. to replace one protocol by a more advanced one. For example, a slow and error-prone modem transmission could be replaced by the latest fibre technology. There is no need to modify the higher layers as long as the service primitives of the lower layers are not changed.

Reusability Two different n-entities may simply use the same $(n-1)$ to 1-services. That is, "re-

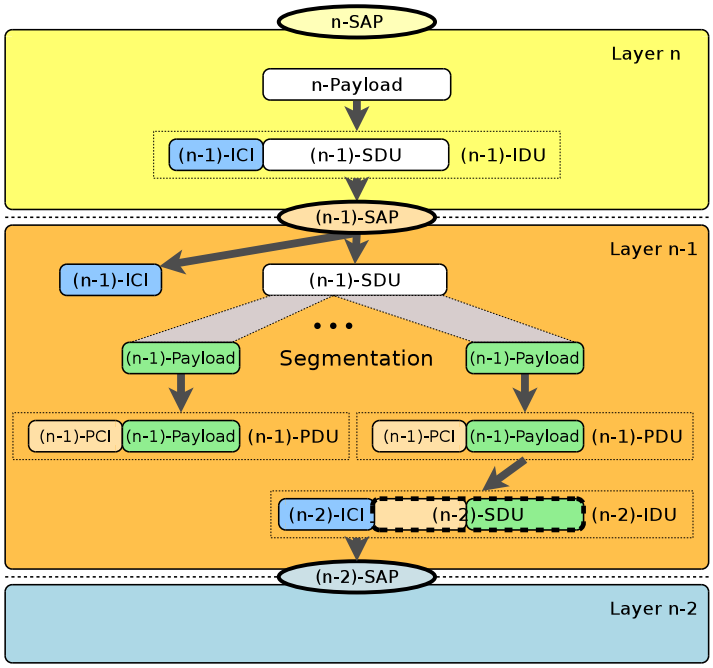


Figure 2.3: The Interface to a Service

inventing the wheel" for the $(n-1)$-service down to the 1-service becomes unnecessary. The same underlying services may be used by different n-entities.

2.2.2 Services

The n-entity does not need any knowledge about the $(n-1)$-protocol that is used by its underlying layer $(n-1)$. Instead, it uses the so-called $(n-1)$-*Interface* of the underlying $(n-1)$-entity, as illustrated in Figure 2.3. That is, it generates a so-called $(n-1)$-*Interface Data Unit* (IDU), which is then provided to the $(n-1)$-entity via a so-called $(n-1)$-*Service Access Point* (SAP). The IDU consists of

1. the n-payload data (e.g. the contents of a complete image file), which is denoted as the $(n-1)$-*Service Data Unit* (SDU), as well as
2. some information about what the $(n-1)$-entity should do with the $(n-1)$-SDU (e.g. "completely transfer the payload data to the peer entity, without reordering or losing information"), which is denoted as the $(n-1)$-*Interface Control Information* (ICI).

The $(n-1)$-entity removes the $(n-1)$-ICI from the $(n-1)$-IDU and processes the $(n-1)$-SDU, according to the instructions provided by the $(n-1)$-ICI. Possibly, one or more $(n-1)$-*Protocol Data Units* (PDU) are generated from the $(n-1)$-SDU. An $(n-1)$-PDU contains

1. data generated from the $(n-1)$-SDU (e.g. a fraction of the $(n-1)$-SDU, which is split up into multiple smaller pieces), as well as
2. information how to re-create the $(n-1)$-SDU from the $(n-1)$-PDUs (e.g. an offset of the contained data) as so-called $(n-1)$-*Protocol Control Information* (PCI).

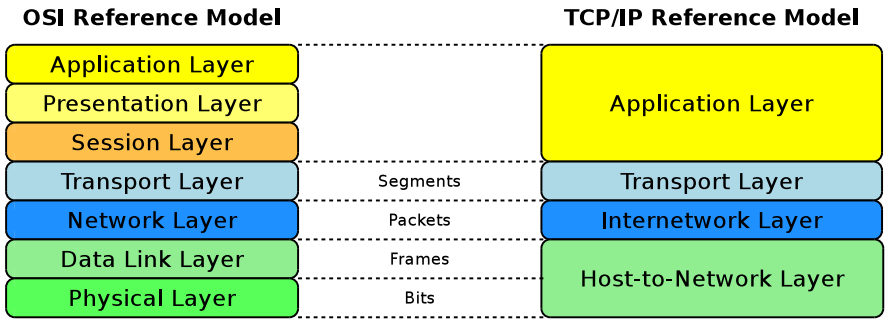


Figure 2.4: The OSI and TCP/IP Reference Models

The splitting of an $(n-1)$-SDU into multiple $(n-1)$-PDUs is denoted as *Segmentation*; the reverse direction is denoted as *Reassembly*. Both will be explained in detail in Section 2.7.

An $(n-1)$-PDU becomes the payload for layer $(n-2)$, i.e. it is combined with an $(n-2)$-ICI to an $(n-2)$-IDU and provided to the $(n-2)$-entity via an $(n-2)$-SAP. After transmission over the physical medium on the bottom of the hierarchy, the data takes the reverse path on the peer protocol stack. Therefore, a further explanation is omitted here. A more detailed introduction as well as an illustrative example are provided in [Tan96, Subsection 1.3.1].

In the following, the set of all n-PDUs belonging to the same communication, e.g. an individual file transfer between two nodes, is denoted as n-*Flow*. If the layer is irrelevant in the described context, it is just denoted as *Flow*.

2.2.3 Reference Models

While a layered protocol stack in principle allows to split up communication tasks into an arbitrary number of layers, two reference models have shown their usefulness in practise and are relevant for this thesis:

- the seven-layered *Open Systems Interconnection Reference Model* (OSI Reference Model) as well as
- the four-layered *TCP/IP Reference Model*.

Figure 2.4 illustrates the models including their layer counterparts as well as their common PDU naming. Both models will be briefly introduced in the following. A more detailed description can e.g. be found in [Tan96, Section 1.4].

2.2.3.1 The OSI Reference Model

The OSI reference model has been developed and initially published by the *International Organisation for Standardisation*[1] (ISO). Later, the publication task has been taken over by the International Telecommunication Union. The latest version of the OSI reference model can be found in [ITU94, Chapter 6]. It consists of seven layers which have the following responsibilities:

Physical Layer: This layer handles the physical transmission of data over a certain physical medium. Particularly, it defines how to transmit bits via e.g. a cable or a wireless channel.

[1] ISO: http://www.iso.org/.

Data Link Layer: The second layer is responsible for the transfer of data between nodes. This includes physical addressing as well as data framing and error correction. The PDU of the Data Link Layer is denoted as *Frame*.

Network Layer: Functionalities for transferring variable-length data sequences from a source to a destination endpoint via one or more networks are duties of this layer. In particular, they include a logical, hierarchical addressing scheme and network routing. The PDU of the Network Layer is called *Packet*.

Transport Layer: This layer takes care of the transfer of user data and includes the segmentation and reassembly of large data blocks, ordered delivery and reliable transfer as well as flow and congestion control. These mechanisms will be introduced in detail later. The payload data unit of the Transport Layer is denoted as *Segment*. It is usually a byte or a datagram.

Session Layer: Dialogue control between end-user applications is the duty of this layer. This includes mechanisms for duplex or half-duplex operation and the definition of checkpoints, adjournment and restart of a session. Particularly, this also includes failover handling in high-availability systems (see also [DR09, Dre07]).

Presentation Layer: The Presentation Layer is responsible for translating data encodings between different systems, e.g. the conversion between different character sets like ASCII[2] and UTF-8[3]. In particular, the Presentation Layer is also responsible for encryption and decryption as well as compression and decompression of data.

Application Layer: This layer is responsible for the actual service of the end-user application, e.g. the reliable transfer of files.

The seven-layered model is sometimes extended by two more layers: the Financial Layer (layer 8) and the Political Layer (layer 9). While this can mainly be seen as humour, it is not too far away from reality. In some other literature, layer 8 defines the User Layer, i.e. problems caused by the human user are often denoted as “layer 8 problems”.

2.2.3.2 The TCP/IP Reference Model

The TCP/IP reference model is based on the historic roots of the TCP/IP-based Internet, which are described in [CK74]. The layering has later been clarified in [Bra89, Subsubsection 1.1.3]. In comparison to the OSI reference model, the TCP/IP reference model is a simplification from seven to four layers. These four layers have the following responsibilities:

Host-to-Network Layer: This layer combines the functionalities of the Physical and Data Link Layers in the OSI reference model. That is, it includes the physical transmission as well as the controlled access to a transport medium.

Internetwork Layer: The Internetwork Layer corresponds to the Network Layer of the OSI reference model. Therefore, each Internetwork Layer protocol can also be seen as a Network Layer protocol.

[2] American Standard Code for Information Interchange, see also [Fis00].
[3] Universal Character Set Transformation Format – 8-bit, see also [Yer03].

Transport Layer: The Transport Layer of the TCP/IP reference model directly maps to the corresponding layer of the OSI reference model.

Application Layer: The Application Layer of the TCP/IP reference model combines the functionalities of the Session, Presentation and Application Layers in the OSI reference model. It is important to note here that a unique mapping of Application Layer protocols from the TCP/IP reference model to the OSI reference model is not possible. For example, the File Transfer Protocol (defined in [PR85]; to be introduced in Subsection 2.13.4) provides the conversion of character encodings. This may be seen as a Presentation Layer functionality. As a result, mappings in literature vary.

To overcome the ambiguity of the Application Layer mappings described above, the following convention is used throughout this thesis: all protocols directly interacting with the service provided for the user are mapped to the Application Layer of the OSI reference model.

2.2.4 Beyond the Layered Protocol Stack

Although the layered protocol stack already provides significant advantages over a naïve "everything in one protocol" approach, the ongoing development of the Internet towards the so-called *Future Internet* shows its limits. For practical reasons, as described in more detail in [Ros06, BM02], it is not always possible to clearly separate a service into independent layers. Instead, the approach of so-called *Cross-Layer Optimisation* introduces inter-layer interaction to provide more efficient data communication services. Details on cross-layer optimisation can be found in [SM05]. Of course, this approach makes an adaptation of the protocol stack by replacing individual layers significantly more difficult.

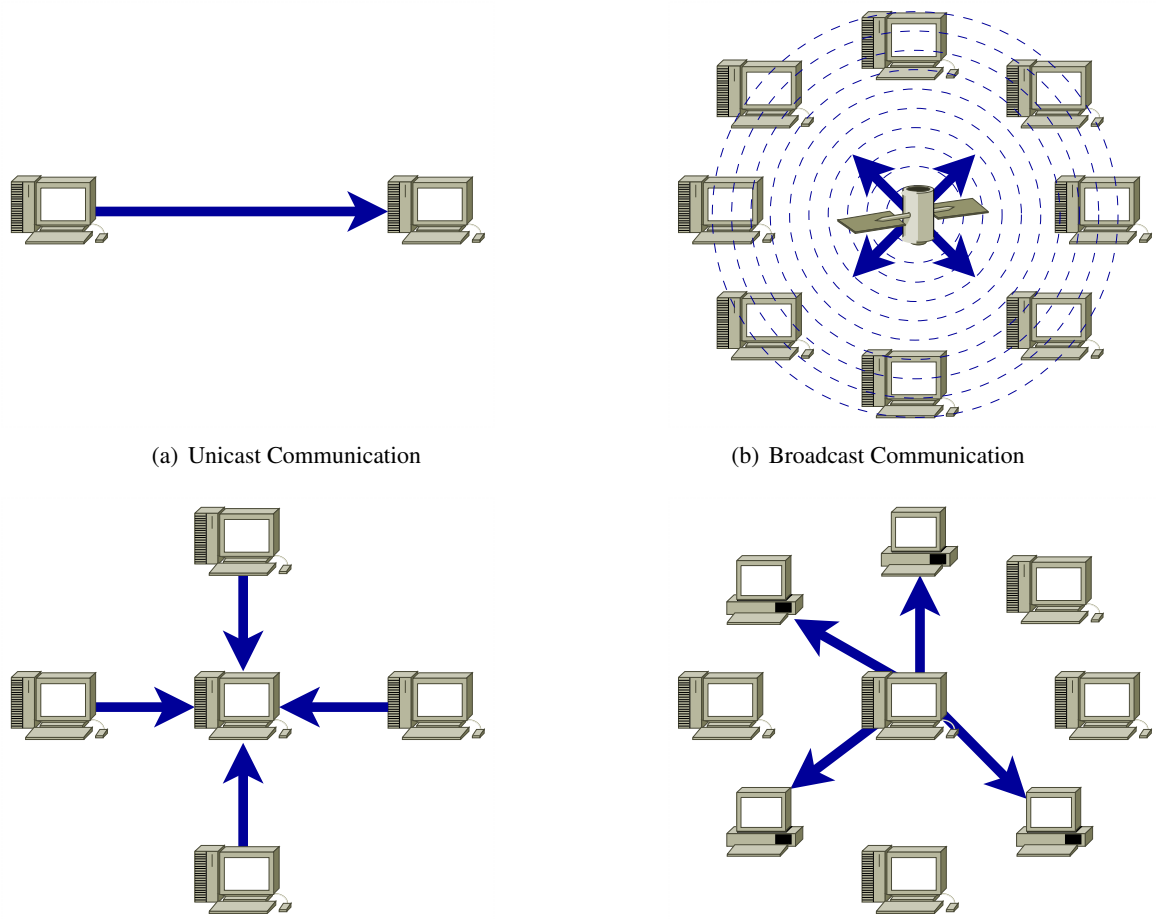
Therefore, the focus of current research on Future Internet is to get rid of the layered protocol stack altogether, and realise services as a composition of loosely coupled, non-hierarchical *Functional Building Blocks*. Such so-called *Clean Slate* approaches are e.g. proposed by [DRB+07, BFH03]. However, since the SCTP protocol takes a more evolutionary than revolutionary path of further Internet development (as described in [DR08c]), the current "work in progress" research ideas and approaches for the Future Internet are not introduced in more detail here. Nevertheless, the general ideas and insights of this thesis also apply to these approaches, as discussed in [DBHR10].

2.3 Classification of Data Communication Services

According to [Bla07], *Data Communications* denotes "the transfer of data or information between a source and a receiver node [within a network]. The source transmits the data and the receiver receives it". That is, the term *Data Communication Service* describes a service which transfers payload data among the entities participating in the communication, by using an appropriate protocol. However, this definition is rather generic. For practical reasons, it is therefore useful to further classify data communication services into subtypes.

2.3.1 Data Communication Services by Participating Entities

Clearly, the most obvious approach of classification is by the entities that are participating in the data transfer. The four types are illustrated in Figure 2.5. Note, that for the classification by participating entities, only the numbers of senders and receivers – but *not* the direction of the communication – are relevant.



(a) Unicast Communication

(b) Broadcast Communication

(c) Concast Communication

(d) Multicast Communication

Figure 2.5: Classification of Data Communication Services by Participating Entities

Unicast Communication denotes the most basic type: a 1:1 communication between one sender and one receiver, as illustrated in Subfigure 2.5(a). Since the SCTP protocol uses this type of communication, it will be explained in more detail below. An example for a unicast communication is e.g. the transfer of a file between two nodes.

Broadcast Communication is a 1:* communication between one sender and an unrestricted number of receivers, as shown in Subfigure 2.5(b). In particular, the sender has no information about the number and the identities of its receivers. The straightforward example for this kind of communication is free-to-air satellite radio and TV transmission.

Concast Communication describes an m:1 communication between m senders and one receiver, as depicted in Subfigure 2.5(c). An example for concast communication is e.g. the transfer of status information to a central monitoring component (such an application is introduced in Subsection B.3.4).

Multicast Communication describes a 1:n communication between one sender and n receivers, as presented in Subfigure 2.5(d). In contrast to broadcast communications – where the set of re-

ceivers is unrestricted – the multicast receivers belong to a well-defined group. That is, the receivers need to explicitly join a group in order to receive the corresponding transmission. An example for this kind of communication is pay-TV, where the set of receivers is restricted to subscribers in possession of the required decryption component.
A special form of multicast communication is so-called *Multipeer Communication*, which describes an m:n communication between m senders and n receivers. Multipeer communication is e.g. used for video conferences, where each participant receives data from as well as sends data to all other participants (see also [Ste00, Chapter 17]).

It should be noted that sender as well as (one of the) receivers may be situated on the same node. Such a kind of node-local communication within a node itself is commonly denoted as *Loopback Communication*.

2.3.2 Data Communication Services by Transfer Directions

The second approach to classify data communication services is by their data transfer directions:

Unidirectional Communication transmits data in just one direction, i.e. from a sender node A to a receiver node B within their network Γ. The direction remains fixed throughout the whole communication. Typical examples of unidirectional communication are satellite radio and TV transmissions. For unidirectional communication, only a trail $\tau_{A\to B} \in \overline{T}_\Gamma(A, B)$ from the sender A to the receiver B is required.

Bidirectional Communication on the other hand transmits data between two nodes A and B within their network Γ in *both* directions, i.e. A and B are senders as well as receivers. Clearly, almost all Internet applications – e.g. requesting and transferring a file – use this kind of communication. For bidirectional communication, two trails are required: a trail $\tau_{A\to B} \in \overline{T}_\Gamma(A, B)$ for the direction from node A to B, as well as a return trail $\tau_{B\to A} \in \overline{T}_\Gamma(B, A)$ back from node B to A.

2.3.3 Data Communication Services by Transferred Data Units

A classification by the transferred data units is the third approach to distinguish different kinds of data communication services. Clearly, the *Bit* (Binary Digit) – with its two distinct states 0 (zero) and 1 (one) – is the canonical data unit of all data communications. However, for practical reasons, all relevant modern communication systems group eight bits to a *Byte*, which is often also denoted as *Octet*. That is, they transmit a multiple of eight bits – which implies an integer number of bytes. Groups of bytes may further be assembled to larger groups, which are denoted as *Datagrams*. In particular, each datagram consists of a well-defined begin and end.

Two data communication variants with respect to the transferred data units are relevant:

Stream-Oriented Communication denotes the transfer of a byte stream, i.e. an arbitrary sequence of bytes. The participating entities have no notion of grouping these bytes to any larger unit. Any grouping is completely in the responsibility of the service user which sends and receives such byte streams. For example, the TCP protocol (defined in [Pos81c]) provides stream-oriented communication.

Datagram-Oriented Communication denotes the transfer of full datagrams. Particularly, begin and end of each datagram are preserved during transfer. That is, the transfer service may *not* split or

join datagrams. If segmentation is necessary, a proper reassembly is required before delivering the datagram to the upper layer of the peer entity. A protocol example providing datagram-oriented communication is the IP protocol (defined in [Pos81b]).

Clearly, it is possible to provide a datagram-oriented data communication service over a stream-oriented data communication service by adding additional control information on beginnings and ends of datagrams at the sender node, as well as processing this information at the receiver node. An example for this way of communication is the ZModem protocol defined in [For88], which provides a datagram-oriented data communication service over the stream-oriented data communication service provided by modems in the Plain Old Telephone System (POTS) network.

Obviously, providing a stream-oriented data communication service over a datagram-oriented data communication service is trivial: the receiver node can just ignore the beginning and the end of each datagram. This is applied for all TCP over IP communications.

2.3.4 Data Communication Services by Transfer Arrangement Procedures

The fourth approach of classifying data communication services is by the arrangement procedures for a payload data transfer:

Connection-Oriented Communication requires the establishment of a so-called *Connection* before – and its release after – any transfer of payload data among entities. A connection denotes the successful completion of necessary arrangements to transfer payload data. All payload data is then associated with its connection. This property can e.g. be used to ensure that SDUs are provided to the service user on the peer side in the same order as they came from the service user on the local side (to be described in detail in Section 2.7). That is, connection-oriented communication consists of three phases (see also [MP93]):

1. Connection establishment,
2. Payload data transfer and
3. Connection release.

For example, a connection-oriented communication service is provided by the POTS: in order to talk to a peer person, it is first necessary to establish a call by dialling. After the talk, the connection is released by hanging up.

Connection-Less Communication on the other hand does *not* need a connection to transfer payload data. An example for a connection-less communication service is the postal system: letters may be sent and received without letter-specific transfer arrangements. Since different letters (i.e. PDUs) are handled independently by a connection-less service (since there is no association with a single connection), it is e.g. not possible to ensure an in-sequence delivery to the service user of the peer side if the underlying service does not ensure a reordering-free transfer.

Note the difference between “connection” and “flow”: in the context of this thesis, the term “flow” (see also Subsection 2.2.2) denotes PDUs of an individual communication, regardless of whether it is connection-oriented or connection-less. On the other hand, the term “connection” directly implies the flow of a connection-oriented communication.

Prefix	Symbol	10^n	1000^n	Long-Scale Name	Short-Scale Name
Kilo	K (k)	10^3	1000^1	Thousand	Thousand
Mega	M	10^6	1000^2	Million	Million
Giga	G	10^9	1000^3	Milliard	Billion
Tera	T	10^{12}	1000^4	Billion	Trillion
Peta	P	10^{15}	1000^5	Billiard	Quadrillion
Exa	E	10^{18}	1000^6	Trillion	Quintillion
Zeta	Z	10^{21}	1000^7	Trilliard	Sextillion
Yotta	Y	10^{24}	1000^8	Quadrillion	Septillion

Table 2.1: The SI Decimal Prefixes and Symbols

2.4 Quality of Service

Since the SCTP protocol only supports unicast communication, details of broadcast, concast and multicast communication services are neglected here. An introduction to such services can e.g. be found in [Ste00, CHW98]. Instead, the focus is on unicast communication in the following.

The term *Quality of Service* (QoS) denotes the capability of a service to provide a "better" service to selected flows. Of course, the meaning of "better" is strongly dependent on the service itself. An application providing video streaming is e.g. concerned about the users' perceptual quality of the playback (see also [RRAW98, Dre01]). Since the focus of this thesis is on the services provided by the Network and Transport Layers (i.e. "lower-layer QoS"), service quality can be reduced to the following four QoS properties here:

1. Throughput,
2. Delay,
3. Jitter, as well as
4. Errors.

These QoS properties will be introduced in the following. Further details on "higher-layer QoS" can e.g. be found in [Ste00].

2.4.1 Throughput

2.4.1.1 Units and Ambiguity

Throughput denotes the amount of data units per time unit that is transported over a path in a network. That is, the resulting throughput ρ_{Transfer} (in bit/s) for transferring an amount of σ_{Data} (in bit) during a time interval of t_{Elapsed} (in s) can be calculated as:

$$\rho_{\text{Transfer}} = \frac{\sigma_{\text{Data}}}{t_{\text{Elapsed}}}.$$

Some more information on measuring throughput is provided in [MA01]; this document is part of the IP Performance Metrics (IPPM) framework introduced in [PAMM98].

Prefix	Symbol	2^n	1024^n	$\approx 10^n$
Kibi	Ki	2^{10}	1024^1	$> 10^3$
Mebi	Mi	2^{20}	1024^2	$> 10^6$
Gibi	Gi	2^{30}	1024^3	$> 10^9$
Tebi	Ti	2^{40}	1024^4	$> 10^{12}$
Pebi	Pi	2^{50}	1024^5	$> 10^{15}$
Exbi	Ei	2^{60}	1024^6	$> 10^{18}$
Zebi	Zi	2^{70}	1024^7	$> 10^{21}$
Yobi	Yi	2^{80}	1024^8	$> 10^{24}$

Table 2.2: The IEC Binary Prefixes and Symbols

Throughput is also commonly denoted as *Data Rate* or *Digital Bandwidth*[4]. It is – usually – measured in bits per second (bit/s). For practical reasons, throughput values are written using the Système International d'Unités (SI) prefixes and symbols defined by the *Bureau International des Poids et Mesures*[5] (BIPM) in [BIP06], in order to shorten their presentation (e.g. 1 Gbit/s instead of 1,000,000,000 bit/s). These standard prefixes are summarised in Table 2.1. In some literature, throughput may also be presented in text form (e.g. "one billion bits per second"). This leads to ambiguity, due to the fact that "one billion" denotes 10^{12} in the so-called *Long-Scale Notation* (which is e.g. used in the Commonwealth and Europe), but only 10^9 in the so-called *Short-Scale Notation* (mainly used in the U.S.A.). In order to avoid this ambiguity, the throughput presentations throughout this thesis consequently apply the SI prefix notation.

Since data communication services above the Physical Layer usually work with full bytes, it is practical to present the *Size* of data in bytes (symbol: B) instead of bits. Furthermore, it is useful to work with data sizes using an integer power of two bytes. Since the SI prefixes define decimal units (i.e. power of 10), they are impractical to represent byte sizes. Therefore, the *International Electrotechnical Commission*[6] (IEC) prefixes and symbols (see [NIS07] and [BIP06, Section 3.1]) – which are shown in Table 2.2 – introduce appropriate *binary* prefixes (e.g. kibibyte for 1,024 bytes). These definitions are relatively new. Therefore, a large amount of literature still uses the ambiguous convention of combining SI prefixes with bytes (e.g. kilobyte for 1,024 bytes). In this thesis, ambiguity is avoided by consequently applying the new IEC prefixes and symbols (e.g. 1 KiB for 1,024 bytes).

2.4.1.2 Overhead and Efficiency

As explained in Subsection 2.2.2, an n-entity combines n-payload with an n-PCI to an n-PDU. From a bit or byte perspective, this means adding additional data – the n-PCI – to the n-payload. This additional data is called *Overhead*. The overhead can be placed before the payload (in this case it is denoted as *Header*) and/or placed after the payload (in this case it is denoted as *Trailer*). An example is depicted in Figure 2.6. Here, layer $(n+1)$ and layer n add headers as well as trailers, while layer $(n-1)$ only adds a header. Regardless of its placement, the overhead reduces the throughput of

[4] In contrast, *Analogue Bandwidth* – measured in Hertz – provides the difference between the upper and lower frequencies of a contiguous frequency band. See also [Tan96, Section 2.1.2].

[5] BIPM: http://www.bipm.org/.

[6] IEC: http://www.iec.ch/.

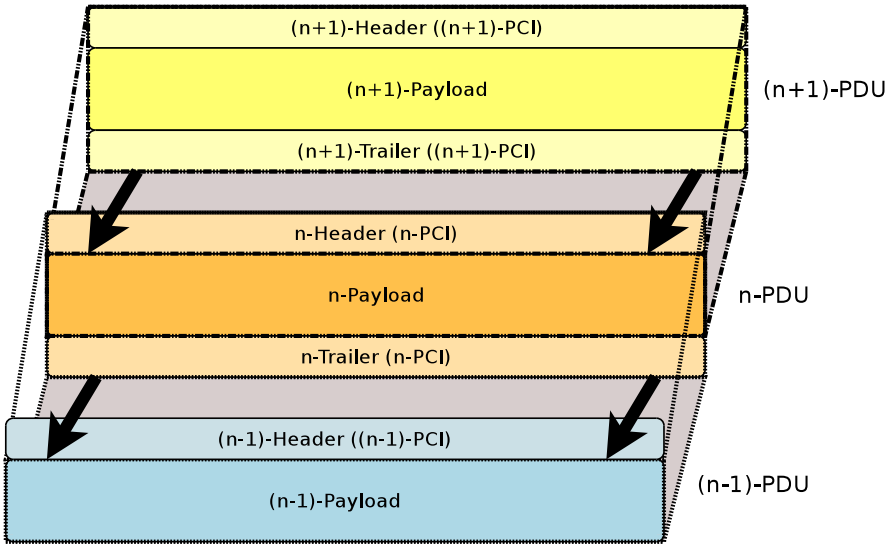


Figure 2.6: Payload as well as Overhead by Headers and Trailers

	Approximated Propagation Speed $v_{\text{Propagation}}$		
Physical Medium	$\times c_0$	m/s	km/h
Thick Coax	0.77	$2.308 * 10^8$	$0.831 * 10^9$
Thin Coax	0.65	$1.948 * 10^8$	$0.702 * 10^9$
Twisted Pair	0.59	$1,769 * 10^8$	$0.637 * 10^9$
Fibre	0.66	$1,979 * 10^8$	$0.712 * 10^9$
Radio Wave (in air)	0.9997	$2,997 * 10^8$	$1.079 * 10^9$

Table 2.3: Approximated Propagation Delays of Physical Media

a service. The *Efficiency* of a data communication service on layer n is therefore defined as:

$$\text{Efficiency}_n = \frac{\text{Payload}_n}{\underbrace{[\text{Payload}_n + \underbrace{[\text{Header}_n + \text{Trailer}_n]}_{\text{Size of } n\text{-PCI}}]}_{\text{Size of } n\text{-PDU}}} \leq 1.$$

The efficiency of a service cannot exceed 1. Clearly, a protocol realising a service should try to maximise the efficiency by keeping the overhead as small as possible.

2.4.2 Delay

Every data transfer takes time, which is denoted as *Latency* or *Delay*. The actual delay δ_{Total} of a data transfer between two nodes consists of five components:

$$\delta_{\text{Total}} = \delta_{\text{Bundling}} + \delta_{\text{Propagation}} + \delta_{\text{Transmission}} + \delta_{\text{Processing}} + \delta_{\text{Queuing}}.$$

The meaning of these five components is as follows:

Bundling Delay: A datagram-oriented service (see Subsection 2.3.3) bundles bytes to datagrams, in order to improve efficiency by decreasing the per-byte overhead (see Subsubsection 2.4.1.2). That is, some data may have to be collected before it can be transmitted. Having a data generation rate of $\rho_{\text{Generation}}$ (in bit/s) and a bundle size of σ_{Bundle} (in bit), the resulting bundling delay (in s) can be computed as:

$$\delta_{\text{Bundling}} = \frac{\sigma_{\text{Bundle}}}{\rho_{\text{Generation}}}.$$

For example, the bundling of an audio stream in Compact Disc quality (i.e. 44100 Hz sampling rate, 16 bits per sample, stereo) into datagrams of 1,452 bytes leads to a bundling delay of $\delta^{\text{Audio}}_{\text{Bundling}} \approx 8.2$ ms.

Propagation Delay: The propagation delay $\delta_{\text{Propagation}}$ denotes the time required for the signal to travel between two nodes. On the Physical Layer, it is:

$$\delta_{\text{Propagation}} = \frac{l_{\text{Medium}}}{v_{\text{Propagation}}},$$

where l_{Medium} denotes the length of the physical medium (in m) and $v_{\text{Propagation}}$ the propagation speed (in m/s). An overview of typical propagation speeds is presented in Table 2.3: for wireless and wired transmissions (speed values based on [Mes05]), it is in the same order of magnitude as the speed of light in vacuum $c_0 = 299,792,458\ \text{m/s} \approx 3 * 10^8\ \text{m/s}$, as defined in [BIP06, Table 7]. That is, $\delta_{\text{Propagation}}$ is in the range of ms for realistic communication distances on earth, e.g. $\delta^{\text{Fibre}}_{\text{Propagation}} \approx 51$ ms for 10,000 km over fibre cable.

Transmission Delay: The delay $\delta_{\text{Transmission}}$ for actually transmitting an n-PDU depends on the size of the n-PDU $\sigma_{n-\text{PDU}}$ as well as the $(n-1)$-SDU throughput of the underlying $(n-1)$-service $\rho_{(n-1)-\text{Service}}$:

$$\delta_{\text{Transmission}} = \frac{\sigma_{n-\text{PDU}}}{\rho_{(n-1)-\text{Service}}}.$$

For example, the transmission delay for transmitting a PDU of 1,500 bytes over a 100 Mbit/s channel is $\delta^{100}_{\text{Transmission}} = 0.12$ ms.

Processing Delay: Nodes also introduce delays for processing data: they need to decode it and decide what to do with it, e.g. to forward it to another node or to provide it to an application on the node itself. This delay is denoted as processing delay $\delta_{\text{Processing}}$. It is usually in the range of µs to a few ms.

Queuing Delay: If it is temporarily not possible to transmit data, e.g. when the underlying physical medium is in use by another transfer, the data is usually stored in a queue. Various types of queues – denoted as *Queuing Disciplines* (QDisc) – are available. Regardless of the applied QDisc, the waiting time in a queue adds a queuing delay δ_{Queuing} to the data transfer.

Clearly, the total *One-Way End-to-End Delay* of a unidirectional data transfer between two endpoints A and B is the sum of all per-hop delays on the used trail. Its measurement within the endpoints is challenging, since it requires their clocks to be accurately synchronised. More details on the measurement of the one-way end-to-end delay are described in [AKZ99a].

For bidirectional transmission, the so-called *Round Trip Time* sums up the end-to-end delays from Endpoint A to Endpoint B, as well as the backwards direction from Endpoint B to A. The RTT

provides a lower bound for the *Response Time* of a service: to get an answer for a request to a remote endpoint, it takes at least one RTT to send the request and receive a response. In comparison to the one-way end-to-end delay, a measurement of the RTT within an endpoint is relatively easy: an endpoint just has to take the difference of answer reception time stamp and request sending time stamp. See also [AKZ99c] for some more details on measuring the RTT.

The choice of a QDisc and its parameters are important configuration settings. Various QDiscs are available; an overview is provided in [Bro06]. In the context of this thesis, two QDiscs are relevant:

First In First Out (FIFO) is the simplest QDisc; it applies the "first come, first served" principle: new data is added to the tail of the queue and data for transmission is taken from its head. As long as the queue does not exceed its size limit, no data is lost. Otherwise, any further data is discarded.

Random Early Detection (RED) introduced by [FJ93] extends the FIFO principle by introducing the two thresholds *MinTh* and *MaxTh*: whenever the average fill level of the queue is between these two thresholds, a packet to be newly enqueued is dropped with a linearly increasing probability from 0 to *MaxP*. For an average fill level exceeding MaxTh, all new packets are dropped. In combination with window-based congestion control – which will be introduced in Subsection 2.11.2 – RED improves the throughput of flows by desynchronising concurrent senders. More details on this subject are provided by [BCC+98]. Guidelines for the configuration of the RED parameters can be found in [Flo97].

2.4.3 Jitter

Jitter denotes the delay variation. For interactive multimedia applications, the jitter is important for the appropriate configuration of playback buffers. These buffers at receiver endpoints are necessary to store data which arrives too early. Also, they are necessary to avoid that a delayed reception of data disrupts the playback. Clearly, the larger the playback buffer, the higher the total delay between sending the data and actually using it for playback. This may lead to a poor user perception.

Details on measuring jitter can be found in [DC02]. A frequently used metric for the jitter is defined in [SCFJ03, Subsection A.8]. It is applied e.g. for Voice over IP (VoIP), which is a highly delay-sensitive and therefore jitter-critical application (see [KR08, Subsection 6.1.2]).

2.4.4 Errors

During transfer over a network, multiple error situations may occur:

Reordering denotes a change of the data sequence during transfer. Reordering occurs when an overtaking of datagrams is possible in a datagram-oriented service, e.g. if using varying trails for transferring different PDUs. Further information on reordering can be found in [MCR+06].

Duplication means that data within the network is duplicated. This can happen due to misconfiguration or malfunction of nodes. [Uij09] explains the measurement of duplication in more detail.

Corruption denotes the distortion of data, e.g. due to transmission problems on the physical medium. It frequently occurs for wireless transmission, due to signal attenuation or interferences.

Loss refers to the loss of data, i.e. sent data that never gets delivered at the receiver node. It may happen due to congestion (i.e. overload) in the network. [AKZ99b] provides some details on measuring losses.

2.4.5 Guaranteed Services and Best Effort

In order to guarantee that certain QoS properties (throughput, delay, jitter and errors rates) are met for PDUs when using a transfer service over a network, mechanisms to reserve forwarding resources on the nodes are necessary. Three variants are possible:

Integrated Services (IntServ) reserves resources for individual flows. In the Internet, IntServ can be realised by the Resource ReSerVation Protocol (RSVP), which provides two types of per-flow reservations: *Controlled Load* (defined in [Wro97]) for reserving bandwidth, as well as *Guaranteed QoS* (defined in [SPG97]) to guarantee a certain bandwidth and a maximum delay. Since RSVP requires per-flow state information within the network, it lacks scalability. This makes a large-scale deployment difficult. More details about this problem can be found in [MBB+97, Subsection 2.1]. Some ideas on efficiently identifying flows in a network are described in [Dre12a]. A detailed introduction on IntServ and RSVP is provided in [Ste00].

Differentiated Services (DiffServ) does not reserve resources for individual flows. Instead, the resources are reserved and guaranteed for a small set of service classes. In the context of DiffServ in IP networks (see [BCD+98] for details), the number of service classes is limited to 64 (due to 6 bits in the IPv4 and IPv6 headers for class identification; see [NBBB98]). That is, individual flows have to be mapped to one of the service classes; guarantees are only provided for these aggregates of flows. Two types of service classes have been standardised in form of so-called *Per-Hop Behaviours* (PHB) for the Internet:

- *Expedited Forwarding* (EF; see [DCB+02]), which can be applied for fixed-bandwidth with low delay and low jitter services (e.g. for interactive real-time applications), as well as
- *Assured Forwarding* (AF; see [HBWW99, Gro02]), which defines a set of four service classes. Each AF class contains three drop precedences. AF allows to overload a class temporarily. But in case of congestion, the drop probability of a PDU increases with its drop precedence. Therefore, AF can be applied for real-time applications with some loss-tolerance. An overloaded AF class may be allowed to borrow resources from an underutilised class. Analogously, an underutilised class may share its temporarily unused resources with other classes.

Note, that PHBs only provide an informal description of a service class. Their actual implementation has to be realised by appropriate QDiscs. An application example and further details on using and configuring DiffServ classes can be found in [Dre01, DSV00]; [Ste00] provides a more detailed introduction to DiffServ.

Best Effort reserves no resources. The network service just tries its best to forward PDUs – hence its name "best effort" – but cannot guarantee any QoS. That is, in case of network overload, data may be lost or arbitrarily delayed.

The Network Layer service of the Internet (see [CK74]) – represented by the IPv4 and IPv6 protocols (see [Pos81b, DH98]) – only provides a datagram-oriented, connection-less, bidirectional best effort service. That is, no QoS properties can be ensured for a service on top of it. In particular, reordering, duplication, corruption and loss may occur. The application of IntServ or DiffServ in the Internet is still very rare yet. That is, services based on top of the Network Layer service in the Internet need to implement appropriate mechanisms to cope with the characteristics of this underlying service.

Listing 1 The Internet-16 Checksum Algorithm

```
1  uint16_t calculateInternet16(const void* data, size_t count)
2  {
3     const uint16_t* addr = (const uint16_t*)data;
4     uint32_t        sum  = 0;
5
6     while(count >= sizeof(*addr)) {    // Main calculation loop
7        sum   += *addr++;
8        count -= sizeof(*addr);
9     }
10    if(count > 0) {       // Add left-over byte, if any
11       sum += *(const uint8_t*)addr;
12    }
13    while(sum >> 16) {    // Fold 32-bit sum to 16 bits
14       sum = (sum & 0xFFFF) + (sum >> 16);
15    }
16    return(~sum);
17 }
```

These protocol mechanisms – mainly realised by Transport Layer services – will be introduced in the following.

2.5 Corruption Detection and Correction

One of the elementary protocol mechanisms to cope with errors is the detection and correction of data corruptions, which may occur due to transmission errors like interferences or attenuation.

2.5.1 Checksums

Obviously, in order to handle corrupted data, it is first necessary to detect any corruption. That is, a receiver must be able to detect whether a received PDU contains the same bit sequence as it has been transmitted by its sender. This can be realised by so-called *Checksums*, which are computed over a PDU and transported – as part of the PDU itself – to the receiver side. In case of corruption, the newly computed checksum at the receiver side differs from the value within the PDU. In the context of this thesis, three kinds of checksum algorithms are relevant. These algorithms will be introduced shortly.

2.5.1.1 Internet-16

A very basic checksum algorithm is Internet-16, which is defined in [BBP88, Rij94]. Listing 1 provides a pseudo-code representation of this algorithm:

- Adjacent bytes are paired to 16-bit values. These values are summed up in a 32-bit variable, which stores the 1's complement sum (lines 6 to 9).
- A remaining single byte is also added (lines 10 to 12).
- Since modern computer systems use the 2's complement notation, the 1's complement sum must be computed by adding any overflows into the least significant bits.

Listing 2 The Adler-32 Checksum Algorithm

```
1  #define BASE 65521    // largest prime smaller than 65536
2
3  uint16_t calculateAdler32(const void* data, const size_t count)
4  {
5      uint32_t s1 = 1;
6      uint32_t s2 = 0;
7
8      for(size_t i = 0; i < count; i++) {    // Compute s1 and s2
9          s1 = (s1 + ((const uint8_t*)data)[i]) % BASE;
10         s2 = (s2 + s1) % BASE;
11     }
12     return((s2 << 16) + s1);    // Combine s1 and s2 to 32-bit checksum
13 }
```

Important properties of the Internet-16 algorithm are that it is simple and easy to implement. Furthermore, it is possible to calculate it incrementally. It is therefore used for a number of important Internet protocols (more details on these protocols will follow in Section 2.13). In order to insert a checksum into a PDU, the checksum field within the PDU is set to zero, the checksum is computed over the PDU and finally written into the checksum field. The receiver can re-compute the checksum again, including the field with the sender's checksum. If the result is 0xFFFF (i.e. -0 in 1's complement notation), no corruption has been detected.

The computed 16-bit checksum results in 2^{16}=65,536 possible values. Therefore, when transmitting a lot of data, there is some chance – 1:65,535 – that a corruption remains undetected. Therefore, more advanced checksum algorithms have been developed.

2.5.1.2 Adler-32

A more advanced approach is the Adler-32 checksum algorithm, defined in [DG96, Subsection 8.2]. The corresponding pseudo-code is presented in Listing 2. In fact, the algorithm computes two 16-bit checksums s_1 and s_2, which are finally combined to a 32-bit checksum (line 12). s_1 sums up one (line 5) plus all bytes of the data, modulo 65,521 (which is the largest prime number smaller than 2^{16}; line 9); s_2 sums up the values of s_1 in each step, also modulo 65,521 (line 10). Adler-32 is quite simple and can be computed very efficiently. However, it has a weakness: e.g. for a 128 bytes PDU, the maximum value for s_1 is 128*255=32,640. That is, the 16-bit space is not fully utilised for short messages. In result, corruptions may remain undetected more easily for smaller messages. A more detailed analysis of the Adler-32 weakness can be found in [SSO02].

2.5.1.3 Cyclic Redundancy Check

A widespread approach for stronger corruption detection (overcoming the described weaknesses of Internet-16 and Adler-32) is the *Cyclic Redundancy Check* (CRC). CRC is based on division in the ring of polynomials over the Galois Field[7] GF(2). These polynomials have binary coefficients, i.e. their value is either 0 or 1; arithmetic is performed modulo 2. A PDU consisting of m bytes is

[7] An introduction to the mathematical basics of Galois Fields can be found in [Cam03].

interpreted as a polynomial $M(x)$ of degree $8 * m - 1$:

$$\begin{aligned} M(x) &= \sum_{i=0}^{8*m-1} a_i * x^i \\ &= a_{8*m-1} * x^{8*m-1} + a_{8*m-2} * x^{8*m-2} + \ldots + a_0 * x^0. \end{aligned}$$

The bits of the PDU define the coefficients of $M(x)$. For example, the j-th bit of the k-th byte may correspond to $a_{8*(m-k)-(8-j)}$; bytes numbered from 0 to $m-1$, bits numbered from 0 to 7. The g coefficients in $M(x)$ that correspond to the checksum to be inserted into the PDU are initialised to 0.

The actual checksum is the remainder $R(x)$ of a polynomial division by a so-called *Generator Polynomial* $G(x)$ of degree g:

$$R(x) = (x^g * M(x)) \bmod G(x).$$

Multiplication by x^g ensures that the dividend always has a higher degree than $G(x)$. In result, the remainder $R(x)$ has a degree of $g-1$ or less. The corresponding coefficient bits of the checksum $R(x)$ are mapped back into the PDU, which then represents the polynomial $T(x)$. At the receiver side, $M'(x)$ (representing the original PDU $M(x)$) and $R'(x)$ (representing the checksum $R(x)$) can be extracted from $T(x)$. The PDU is assumed to be uncorrupted, if:

$$(x^g * M'(x)) \bmod G(x) = R'(x).$$

Depending on the choice of $G(x)$ (and in particular its degree g), CRC can reliably detect various types of corruption occurring during transmission. Note, that CRC cannot protect against specially-crafted modification (e.g. in attack scenarios), i.e. CRC is not applicable as cryptographic hash function. A more detailed analysis of different CRC variants can be found in [Koo02, HŘS+06]; a detailed introduction including a numerical example is provided by [Tan96].

Relevant for this thesis is the CRC variant *CRC-32C*, which is defined in [CBH93][8]. CRC-32C uses the following generator polynomial $G_{\mathrm{CRC-32C}}(x)$[9]:

$$\begin{aligned} G_{\mathrm{CRC-32C}}(x) &= x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + \\ &\quad x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^0 \ . \end{aligned}$$

That is, the resulting CRC code has a size of 32 bits. Some details on the implementation of CRC-32C – including code written in C – can be found in [Ste07, Appendix B]. In comparison to Adler-32, an implementation is significantly more complex, leading to increased processing time requirements.

2.5.2 Forward Error Correction

While a checksum can only detect the corruption of a PDU, it is not possible to fix it. The usual procedure therefore is to just drop corrupted PDUs and handle them as being lost (to be explained in Section 2.9). However, in certain application cases, it is not possible to re-transmit the data efficiently. For example, to recover mangled data stored on a harddisk, the user would need to restore a – hopefully existing – backup. In such cases, so-called *Forward Error Correction* (FEC) can be applied. FEC adds redundancy information to PDUs, which makes it possible to correct a limited number of corrupted

[8] The paper by [CBH93] denotes CRC-32C as CRC-32/4. In later literature, e.g. [Ste07, SSO02], it is commonly denoted as CRC-32C, after the first author Castagnoli.

[9] The numerical representation of $G_{\mathrm{CRC-32C}}(x)$ is 0x11EDC6F41.

bits. The higher the possible corruption rate, the more bits are actually needed for redundancy in order to fix a corruption. That is, FEC may significantly increase the overhead.

A more detailed introduction is neglected here, since FEC is not directly relevant in the context of this thesis. Some more information on FEC can be found in [Tan96, Subsection 3.2.1], an overview of FEC algorithms is provided by [MZ08].

2.6 Sequence Numbering

Like corruption detection and correction, the unique enumeration of objects – in form of so-called *Sequence Numbers* – is another common protocol mechanism. A sequence number is an integer counter $q \in \mathbb{N}_0$. Whenever a unique identifier o has to be allocated to a new object, o is set to the value of q and q is incremented by 1. Let $q = 0$ on initialisation. Then, the identifier $o_1 = q = 0$ will be allocated to the first object. After that, $q = 1$. The second object will get the identifier $o_2 = 1$, leading to $q = 2$, etc.. Obviously, all object identifiers are unique, and – since $|\mathbb{N}_0| = \infty$ – a countably infinite number of objects can be enumerated *theoretically*.

In practise, memory is limited and therefore the size of a sequence number is restricted, too. That is, $q \in [0, \ldots, 2^{\mathrm{SeqNumberBits}} - 1] \subset \mathbb{N}_0$, for $\mathrm{SeqNumberBits} \in \mathbb{N}$ an appropriate number of bits. This results in $2^{\mathrm{SeqNumberBits}}$ possibilities to uniquely allocate identifiers. In order to keep $\mathrm{SeqNumberBits}$ small (i.e. to reduce space requirements), it is possible to remember the pool of unallocated identifiers and install a procedure to release allocated identifiers when they are not needed any more. Remembering unallocated identifiers could e.g. be realised by a bit field. However, the allocation and deallocation procedures in this case add additional time and space complexity.

An alternative is therefore to retain the original allocation procedure of mapping the identifier to the current sequence number value and incrementing it. However, this leads – due to the limited value space of q – to an overflow when trying to increment q: $(2^{\mathrm{SeqNumberBits}} - 1) + 1 = 0$. This overflow is denoted as *Counter Wrap*; it may lead to a non-unique identifier allocation. Therefore, it puts a constraint on the sequence number usage: the sequence number user must always ensure that when allocating identifier o_i, there is no more object o_j with $o_j = o_i$. Under this condition, the space-limited sequence number still ensures unique identifiers.

Note, that the choice of $\mathrm{SeqNumberBits}$ limits the identifier allocation rate: given a maximum lifetime $\mathrm{MaxLifetime}$ (in s), the allocation rate $\mathrm{AllocationRate}$ (in objects/s) is limited to

$$\mathrm{AllocationRate} \leq \frac{2^{\mathrm{SeqNumberBits}}}{\mathrm{MaxLifetime}}.$$

For example, $\mathrm{AllocationRate} \leq 1,092$ for $\mathrm{MaxLifetime}$=60 s and $\mathrm{SeqNumberBits}$=16 bits. An inappropriate choice of $\mathrm{SeqNumberBits}$ may lead to a future limitation of the system, e.g. when the allocation rate needs to be increased to meet extended requirements. More details on sequence numbers are e.g. provided by [EB96].

2.7 Segmentation and Reassembly

A further important protocol mechanism – which is usually realised on the Network and/or Transport Layer – is the *Segmentation* and *Reassembly* of n-SDUs, as illustrated in Figure 2.7. Here, the underlying layer $(n-1)$ limits the maximum $(n-1)$-SDU size. This limit is denoted as *Maximum Transmission Unit* (MTU). Therefore, layer n has to carefully craft the n-PDUs, in order to avoid

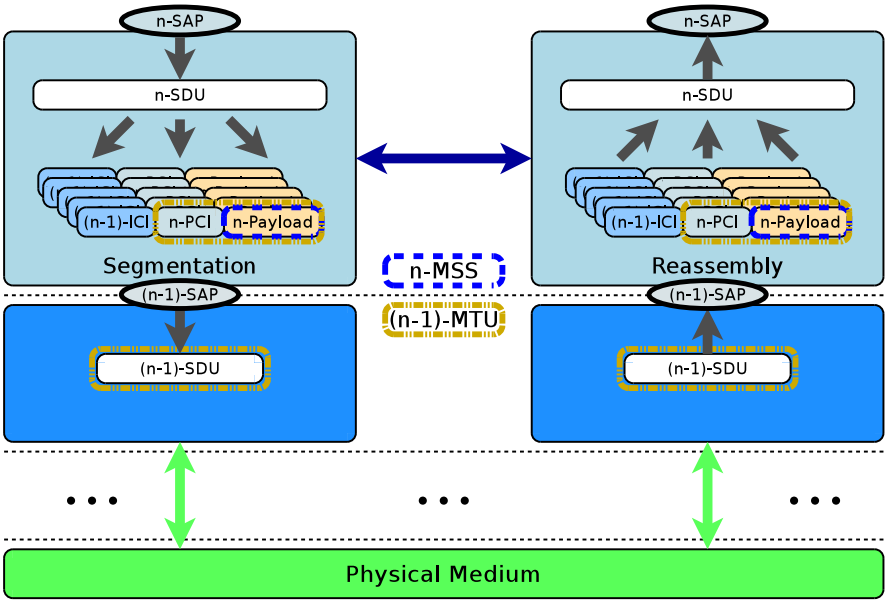


Figure 2.7: Segmentation and Reassembly

exceeding the MTU. That is, the maximum size of the n-payload – which is denoted as *Maximum Segment Size* (MSS) is:

$$\text{MSS}_n = \text{MTU}_{n-1} - \text{Size of PCI}_n.$$

The MTU is usually a compromise between efficiency by reduced overhead (larger MTU is better; see Subsubsection 2.4.1.2) and reduction of data loss on corruption (smaller MTU is better; see Section 2.5). In a network $\Gamma = (L, N, C, c)$, the MTU for a trail $\tau \in \overline{T}_\Gamma(n_1, n_2)$ between two nodes n_1 and n_2 is given by the minimum MTU of any channel used by τ (i.e. "the weakest link in the chain").

The sender n-entity has to split up too-large n-SDUs into appropriately-sized n-PDUs; the peer entity has to put the n-payloads of these n-PDUs together to reconstruct the *payload* of the original n-SDU. Note, that it is *not* necessary to reconstruct the original n-SDU itself for stream-oriented communication. The n-payloads may be delivered in form of multiple n-SDUs, within a larger n-SDU, etc.. The original n-SDU has to be reconstructed for datagram-oriented communication only, in order to preserve begin and end of each datagram (see also Subsection 2.3.3).

The information necessary to reassemble the n-payloads has to be written into the n-PCIs of the n-PDUs. The n-PDUs are commonly denoted as *Segments* (hence the name of the Transport Layer data units) or *Fragments*. In the context of this thesis, the following mechanisms for actually implementing segmentation and reassembly are relevant:

- A segment can store a unique identifier (usually a sequence number; see Section 2.6) of its original n-SDU, as well as the offset – i.e. a byte position – of its payload within the payload of the original n-SDU.

- Alternatively, the segments can be enumerated (by a sequence number; see Section 2.6). Then, the peer entity is aware of their order and can reconstruct their original position within the payload.

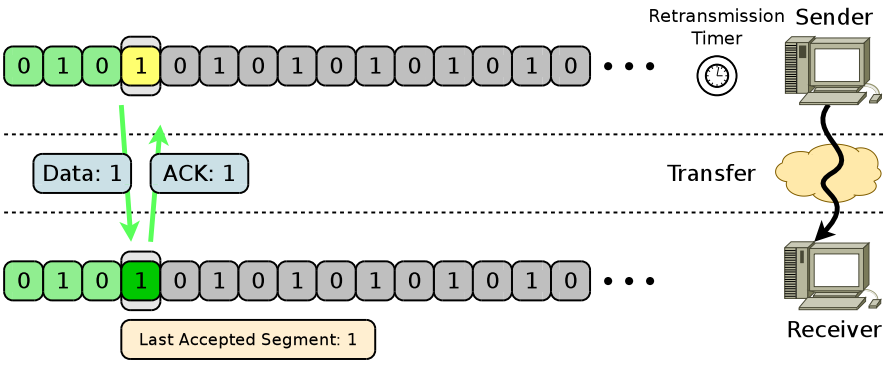


Figure 2.8: The Principle of "Stop and Wait" for Reliable Transfer

- If the first as well as the last segment of a datagram are marked appropriately, it is also possible to reconstruct the original n-SDU (i.e. the original datagram).

2.8 Ordered Delivery

The protocol mechanism of segmentation and reassembly ensures that n-SDUs are transferred completely. However, it does not ensure that the n-SDUs are delivered by the peer entity to its $(n+1)$-entity in the same order they have been provided by the local $(n+1)$-entity. If the n-PDUs are transferred via different trails in the network, they may overtake each other. A delivery which may be out of its original order is denoted as *Unordered Delivery*.

In order to provide a so-called *Ordered Delivery*, i.e. preserving the original order of all segments, sequence numbers for the n-PDUs may be applied – similar to segmentation. However, instead of just ensuring the order within an n-SDU, the order of the whole set of n-SDUs is ensured. For efficiency reasons, segmentation and ordered delivery may be combined by using the same sequence number within the n-PCIs of the n-PDUs (i.e. a reduction of overhead space).

2.9 Reliable Transfer

The ordered delivery protocol mechanism just ensures that the order of the data is preserved. However, losses of n-PDUs result in "gaps" in the data sequence. Such a transport service is denoted as *Unreliable Transfer*. It may e.g. be acceptable for video streaming, where some missing data just results in a temporary artefact (see also [Ste00]). However, a large fraction of services – like e.g. a file transfer – does not tolerate any partial loss of data. These services require a *Reliable Transfer* service, which ensures that no data is lost.

2.9.1 Naïve Approach: Stop and Wait

A naïve approach for a protocol mechanism to realise a reliable transfer is denoted as *Stop and Wait*. Its principle is illustrated in Figure 2.8: segments are enumerated by a sequence number (see Section 2.6). In order to reliably transfer a segment s, the sender sends it as *Data-n-PDU*; its n-PCI includes the sequence number. Furthermore, it starts the *Retransmission Timer*, which counts the time

from a configured *Retransmission Timeout* (RTO) down to zero. When eventually the segment s arrives at the receiver, it generates an *Acknowledgement* (ACK) for segment s, in form of an *Ack-n-PDU* including the sequence number to be acknowledged – here: s – in its n-PCI. This acknowledgement tells the sender about the successful reception of segment s. It can now cancel the retransmission timer and proceed – in the same way – with segment $s + 1$. This is the ideal case, i.e. no loss has occurred.

However, two things could go wrong:

Loss of Data-n-PDU If a Data-n-PDU (i.e. a segment) is lost, the receiver obviously cannot acknowledge it. When the retransmission timer expires, the sender just has to retransmit the Data-n-PDU (i.e. to send it again). This procedure is denoted as *Retransmission* (RTX).

Loss of Ack-n-PDU A lost acknowledgement finally results in the sender assuming the Data-n-PDU as being lost. It has to retransmit it upon expiration of the retransmission timer. The receiver will get a duplicate. However, by remembering the sequence number of the last successfully accepted Data-n-PDU, it detects the duplicate, ignores it, but resends its Ack-n-PDU.

Note, that the sequence number for stop and wait may have the size of a single bit only (i.e. $\mathrm{SeqNumberBits}$=1; see Section 2.6), if the underlying service ensures ordered delivery for the n-PDUs (i.e. the segments). Otherwise, the time between two Counter Wraps must be higher than the maximum lifetime of an n-PDU in the network.

The stop and wait protocol mechanism is already sufficient to provide a reliable transfer service. However, its throughput is limited by the RTT between sender and receiver: the transport of each segment requires at least one RTT (i.e. from starting to send the Data-n-PDU until having received the corresponding Ack-n-PDU). That is, the throughput is at most $\frac{1}{\mathrm{RTT}}$ n-segments/s in the ideal case (i.e. without any loss).

2.9.2 Pipelined Approach: Sliding Window

Pipelining is an approach to overcome the key problem of stop and wait (i.e. having to wait most of the time for an acknowledgement, unless the RTT is extremely small). That is, instead of having at most one segment on travel through the network and not yet acknowledged, which is denoted as *Outstanding* or *In Flight*, multiple segments may be outstanding.

2.9.2.1 Principle

The principle of the so-called *Sliding Window* protocol mechanism, as illustrated in Figure 2.9, is to have a so-called *Send Window* at the n-entity of the sender side. This send window has a size of W_S segments; the larger W_S, the more segments may be outstanding. In the depicted example, W_S=6 segments. This means that after sending segment #42, five more segments (i.e. W_S=6 in total) may be transferred prior to receiving any acknowledgement from the receiver side. Here, segments #42 to #47 have actually been sent. However, although they have been sent, they must remain in the so-called *Send Buffer* until being acknowledged. Two of the eight segments in the send buffer (segment #48 and #49) are still waiting for their transfer. Since only up to W_S=6 outstanding segments are allowed, they may not be sent yet.

Segment #42 as well as segments #44 to #46 have successfully arrived at the n-entity of the receiver side. The receiver n-entity accepts them, because these segments fall into its so-called *Receive Window*. In the example depicted in Figure 2.9, the size of the receive window is W_R=8 segments.

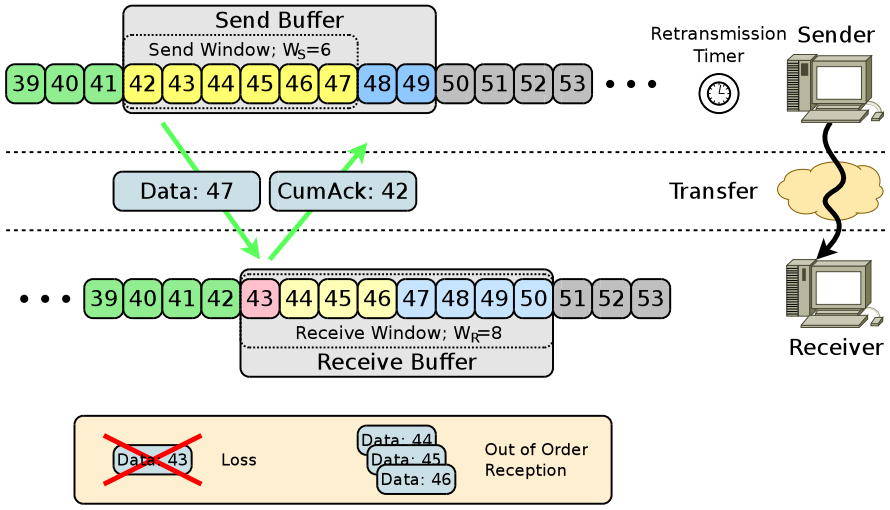


Figure 2.9: The Principle of a "Sliding Window" for Reliable Transfer

Here, the receive window covers the complete space of the *Receive Buffer*, i.e. the place where received segments are stored to ensure ordered delivery. Since segment #42 is the first segment in the receive window, it is advanced to segment #43. That is, all segments until #42 have been successfully received – if such segments are received again, they must be duplicates and will be ignored therefore. The new lowest segment to be accepted is segment #43. Since the receive window size W_R remains constant, the highest segment to be accepted is increased to segment #51. Since segment #43 has been lost, there is a gap in the segment sequence. The receiver therefore sends a so-called *Cumulative Acknowledgement* (CumAck) – in form of a *CumAck-n-PDU* – for segment #42, denoting that all segments including segment #42 have been received successfully.

When the sender n-entity processes this CumAck, it may shift its send window ahead, then having segment #43 as the first segment and segment #48 as the last one. That is, segment #48 may be transferred now. Furthermore, since the send buffer space that had been occupied by segment #42 before has become available, the newly gained space can be filled by segment #50. This new segment may be generated from further n-SDUs of layer $(n+1)$.

Due to the shifting of send and receive windows within the segment sequence number space, the protocol mechanism is denoted as sliding window.

The following error handling procedures have to be applied:

- The n-entity detects the loss of a Data-n-PDU s_i by receiving an out-of-sequence Data-n-PDU s_j with $s_i < s_j$. In this case, it sends a CumAck-n-PDU for the last in-sequence segment.

- The loss of the last sent Data-n-PDU cannot be detected by the receiver side (since the receiver is not aware of its existence). Like for stop and wait, a retransmission timer is necessary to detect this loss at the sender side.

- Due to cumulative acknowledgements, a lost CumAck-n-PDU is superseded by the following one. If the last CumAck-n-PDU of a sequence is lost, the retransmission timer triggers a segment retransmission. The reception of a duplicate segment finally results in sending a new CumAck-n-PDU.

2.9.2.2 Retransmission Strategies

In order to actually retransmit a lost segment, multiple so-called *Retransmission Strategies* are possible and frequently applied (see also Section 2.13):

Go Back N In the simplest case, W_R=1. Then, the reception of each out-of-sequence segment triggers a CumAck-n-PDU for the last successfully received segment being in sequence. That is, the receiver side only accepts in-sequence segments; any out-of-sequence segments are ignored – and must be retransmitted by the sender later (starting from the first missing segment N, hence the name Go-Back N). This mechanism works well when losses are rare, but may become extremely inefficient for an error-prone and high-RTT connection (due to a large number of in fact unnecessary retransmissions).

Selective Repeat Instead of ignoring all out-of-sequence segments (as for Go-Back N), the n-entity of the receiver side may generate CumAcks as before, but including a list of so-called *Gap Acknowledgements* (GapAcks) listing the successfully received segments ahead of the cumulatively acknowledged segment. Such an acknowledgement – containing a CumAck and possibly GapAcks – is denoted as *Selective Acknowledgement* (SACK). Therefore, the corresponding n-PDU will furthermore be denoted as *SelAck-n-PDU*.
In the example shown in Figure 2.9, this acknowledgement would contain a CumAck for segment #42, as well as GapAcks for segments #44 to #46. Then, the sender is able to just skip the retransmission of segments that have been acknowledged by GapAcks. Clearly, for ordered delivery, the receiver must now keep segments #44 to #46 in its receive buffer, until the missing segment #43 has arrived. Then, all segments may be forwarded to the $(n+1)$-entity (in form of n-SDUs). On the other hand, for unordered delivery, the segments may leave the receive buffer immediately.

Cyclic Preventive Retransmission If the RTT is high, it may furthermore be useful for a sender to cyclically start retransmitting the segments in the send window – even without having the information about a segment loss.

In the following, segments that have been acknowledged by a CumAck are denoted as being *CumAck'ed*. Analogously, a *GapAck'ed* segment has been acknowledged by a GapAck.

2.9.2.3 Window Size Constraints

The sizes of send and receive window – W_S and W_R – are limited by the size of the sequence number space used for enumerating the segments. Let $\mathrm{SeqNumberBits}$ be the sequence number size in bits (see also Section 2.6). Then, the following constraints apply:

$$\begin{aligned} W_S &\leq \frac{2^{\mathrm{SeqNumberBits}}}{2}, \\ W_R &\leq \frac{2^{\mathrm{SeqNumberBits}}}{2}. \end{aligned}$$

These limits are necessary to ensure uniqueness of the segment numbers. Let $\mathrm{SeqNumberBits}$=3 (i.e. segment numbers out of $[0, \ldots, 7] \subset \mathbb{N}_0$) and W_R=8 (i.e. $W_R = 2^3$). Then, after having sent the segments from #0 to #7, a CumAck for segment #7 will acknowledge all of them. However, if all of them have been lost, a CumAck for segment #7 would be an acknowledgement of the previous round (telling the sender that its previously sent segments have to be sent again, starting from segment #0).

This ambiguity leads to data loss. The constraint above ensures that the window size is at most half of the sequence number space, ensuring uniqueness of the segments. A setting of $\mathrm{SeqNumberBits} \geq 4$ solves the problem here.

In order to achieve the optimum segment throughput, the minimum size of W_S must at least cover the so-called *Bandwidth-Delay Product* (BDP; see also [CXSN04]), which is given by the segment throughput ($\mathrm{Throughput}_n$; in segments/s) as well as the RTT between sending a segment until processing its acknowledgement (RTT_n; in s):

$$\mathrm{BDP}_n = \mathrm{Throughput}_n * \mathrm{RTT}_n.$$

Note, that "delay" for the BDP means the time from sending a segment until it gets acknowledged, i.e. the RTT (see also Subsection 2.4.2). Therefore, in order to avoid ambiguity, the BDP will be denoted as *Bandwidth-RTT Product* in the context of this thesis. That is, W_S (in segments) must ensure that there can be at least as many segments outstanding as segments can be transmitted at the given throughput ($\mathrm{Throughput}_n$; in segments/s) during one RTT (RTT_n; in s):

$$W_S \geq \underbrace{\mathrm{Throughput}_n * \mathrm{RTT}_n}_{\mathrm{BDP}_n}.$$

This implies that a too-small setting of W_S limits the achievable throughput:

$$\mathrm{Throughput}_n \leq \frac{W_S}{\mathrm{RTT}_n}.$$

An appropriate choice of windows size limits – with respect to future advances in throughput and changes of the RTT – is therefore an important design decision. Details on extending the window sizes of the TCP protocol – which had shown to be too small for fast, intercontinental connections – are explained by [JB88]. A more detailed introduction to the sliding window protocol mechanism, including examples and code listings, can be found in [Tan96, Section 3.4].

2.9.3 Overhead Reduction and Performance Improvements

In order to further reduce the overhead and improve the performance of reliable transfer, multiple mechanisms are commonly used.

2.9.3.1 Delayed Acknowledgement

Since CumAcks are cumulative, a subsequent CumAck supersedes a former CumAck. That is, instead of generating a new SelAck-n-PDU as soon as a Data-n-PDU has been received, a receiver may wait some time for the reception of further Data-n-PDUs. Then, a single CumAck may acknowledge multiple Data-n-PDUs. This optimisation is denoted as *Delayed Acknowledgement* (see also [Bra89]).

2.9.3.2 Piggybacking

For bidirectional transfer, the overhead of acknowledgements may be further reduced by applying so-called *Piggybacking* (see also [Tan96]): instead of sending a separate SelAck-n-PDU (containing a CumAck'ed sequence number and possibly a list of GapAcks in its PCI), the acknowledgement information can be added to the PCI of a Data-n-PDU in reverse direction. That is, instead of having to transfer two separate PDUs (one SelAck-n-PDU and one Data-n-PDU) in reverse direction, just a single combined DataSelAck-n-PDU may be used.

2.9.3.3 Bundling

As described in Subsubsection 2.4.1.2, the efficiency of a data transfer is best when the overhead remains small. That is, for bulk data transfer on layer n, it is useful to fully utilise the n-MSS (see also Section 2.7). If the $(n+1)$-entity generates lots of small n-SDUs, an immediate transfer of the corresponding n-PDUs leads to a low efficiency.

The strategy to improve efficiency is denoted as *Bundling*. At the expense of increased delay, it defers the generation of n-PDUs. Instead, the data of the n-SDUs is collected (in a buffer) until there is either enough data to fill an $(n-1)$-MTU, or a timer – denoted as *Bundling Timer*, which has been initially set to the *Bundling Timeout* – expires. Then, an appropriate n-PDU is generated, including the collected data en bloc. This protocol mechanism is frequently denoted as *Nagle's Algorithm* (see also [Nag84]).

2.9.3.4 Fast Retransmission

In order to avoid waiting for the expiration of the retransmission timer (i.e. waiting up to a full RTO, as explained in Subsection 2.9.1), it is useful to introduce so-called *Fast Retransmission*, as defined in [APB09, Subsection 3.2]: a fast retransmission is triggered when a segment j that has been reported as missing for a certain number of times. Segment j is assumed to be missing if a SelAck-n-PDU contains a CumAck for segment i and a GapAck for a segment k with $i < j < k$. The threshold for the number of such SelAck-n-PDUs is denoted as *Duplicate Threshold* and set to three in [APB09, BAFW03]. When reaching this threshold, the fast retransmission for j is triggered *once*. That is, segment j is retransmitted as soon as possible, and its retransmission timer is reset.

Any further retransmissions are handled – as before – by expiration of the retransmission timer. Therefore, these retransmissions will further be denoted as *Timer-Based Retransmissions*.

Note, that using fast retransmission with delayed acknowledgement (as described in Subsubsection 2.9.3.1) requires to send an acknowledgement *immediately* when receiving an out-of-sequence segment.

2.9.3.5 Handling of Data Corruption

In combination with corruption detection by a checksum mechanism (see Subsection 2.5.1), the reliable transfer mechanism can also be used to handle corruptions: mangled n-PDUs are simply ignored. Then, they will be handled as lost and finally retransmitted.

This mechanism may be improved further by the introduction of a so-called *Negative Acknowledgement* (NAK), which is generated by the receiver upon reception of a corrupted Data-n-PDU. A *Nak-n-PDU* can tell the sender not to wait for the expiration of its retransmission timer, but instead to retransmit the corresponding Data-n-PDU as soon as possible.

Of course, in order to tell the sender which Data-n-PDU has to be retransmitted, the receiver must examine the corrupted PDU. In practise, it can check the required information (particularly, the sequence number) for plausibility and send a Nak-n-PDU when this information seems to be reasonable. In the – very unlikely – case of a corrupted but plausible sequence number, a retransmitted Data-n-PDU becomes a duplicate and will just be ignored.

2.10 Flow Control

When a powerful sender transfers data over a high-throughput network to a low-performance receiver, the receiver may easily become overloaded. That is, it becomes incapable of accepting any new segments (e.g. if the receive buffer is full and data cannot be forwarded quickly enough to the $(n+1)$-entity). In this case, newly incoming segments will get lost. Reliable transfer will – of course – trigger retransmissions and ensure a loss-free communication for layer $(n+1)$. However, unnecessary retransmissions reduce the efficiency. So-called *Flow Control* denotes protocol mechanisms which are responsible of protecting the receiver from overload.

2.10.1 Approaches

In the simplest case, the sender could just use a limited throughput that is known to be acceptable for the receiver. This approach is denoted as *Open-Loop Flow Control*, since the sender does not need any feedback from the receiver. On the other hand, so-called *Closed-Loop Flow Control* requires such feedback. This feedback could e.g. be provided in the form of special n-PDUs to let the receiver tell the sender to temporarily suspend transmission ("Receiver not Ready"), or to resume transmission again ("Receiver Ready"). [Tan96, Subsection 3.6.1] provides a protocol example.

Alternatively, as explained in detail by [Tan96, Subsection 6.2.4], the maximum allowed number of outstanding bytes may be dynamically adapted, in order to throttle the sender. That is, the receiver tells its desired limit to the sender; this size is known for the receiver to work properly (e.g., the receiver has allocated an appropriately-sized buffer to accept the given number of segments). This information may be provided by adding a so-called *Advertised Receiver Window* to the SelAck-n-PDUs. The advertised receiver window $\mathrm{AdvertisedReceiverWindow}_n$ (in segments) contains the amount of available space in the receive buffer, at the time of sending the acknowledgement. The sender can calculate the actual amount of new segments it is allowed to send – denoted as *Peer Receiver Window* $\mathrm{PeerReceiverWindow}_n$ (in segments) – with its knowledge of the amount of currently outstanding segments $\mathrm{Outstanding}_n$ (in segments):

$$\mathrm{PeerReceiverWindow}_n = \mathrm{AdvertisedReceiverWindow}_n - \mathrm{Outstanding}_n.$$

That is, the peer receiver window contains the amount of acceptable segments minus the number of segments being outstanding (i.e. not yet acknowledged, but already on their way).

2.10.2 Silly Window Syndrome

A problem which can occur for window-based flow control is denoted as the *Silly Window Syndrome*. It is described in [Bra89]: when the data from the receive buffer can only be forwarded to the $(n+1)$-entity in small pieces – because of e.g. low processing performance – the advertised receiver window goes down to zero, telling the sender to suspend transferring new segments. When the next piece of data leaves the receive buffer (which could be as small as a single byte), the receiver may advertise this small amount of available space. The sender may then fill it quickly, and the same procedure is repeated. Clearly, this leads to a high overhead and therefore to a low efficiency.

To overcome the Silly Window Syndrome, it is recommended to advertise newly available receive buffer space after suspending the transmission only when an "appropriate" amount of buffer space (e.g. enough to accept at least one full MSS; see Section 2.7) has become available again.

2.10.3 Zero-Window Probing

Suspending the sender by an advertised receiver window of zero leads to a further error case: if the SelAck-n-PDU for resuming the transfer again (by advertising a non-zero window) is lost, the sender may wait indefinitely. Therefore, in case of a suspended transfer, the sender has to perform a regular *Zero-Window Probing*, i.e. to check whether the receiver is able to accept new segments. This could e.g. by realised by trying to send a single new segment, or by introduction of a special n-PDU for probing.

2.11 Congestion Control

While flow control protects the receiver from overload, it does not protect the network. The superposition of the data flows from many senders, or just a powerful sender, may cause overload in a network. This overload is denoted as *Congestion*; its analogue is a traffic jam on a crowded motorway. The protocol mechanism of *Congestion Control* is responsible for avoiding congestion.

2.11.1 Overview of Approaches

An obvious solution to protect a network from overload is to apply IntServ or DiffServ (see also Subsection 2.4.5): forwarding resources (in particular: bandwidth and queue capacities) are explicitly reserved. This approach is e.g. applied by ATM networks, which are described in [RW97]. Details on these congestion control approaches can also be found in [Wel05]. Since the Internet – which is in the focus of this thesis – only provides a best effort service, they are not explained in detail here.

In a network which is only able to provide a best effort service, congestion results in packet losses. In the early days of the Internet, as described in [CK74], congestion control was not realised. The problems related with this effect were first seen in the network of the Ford Motor Company, after adding more and more channels of widely varying throughputs and delays. Increased load has filled up the queues in the routers, increasing RTTs, and finally resulting in retransmission timers expiring before packets actually got delivered at their destination. Research by [Nag84] has shown that this so-called *Congestion Collapse* – once this state has been reached – remains stable.

As a solution to the congestion challenge, [JK88] has proposed the "*Conservation of Packets Principle*": for a flow in equilibrium, i.e. running stably with a full send window of segments in transit, a new segment should not be put into the network until an old segment has left it. That is, new segments should only be transferred after older ones have been acknowledged. Packet losses are assumed to be the result of congestion.

2.11.2 Window-Based Congestion Control

The Internet approach to congestion control is window-based. That is, the sender endpoint maintains a *Congestion Window*[10] variable c (in segments), which denotes the maximum allowed number of outstanding segments. So-called *Additive Increase, Multiplicative Decrease* (AIMD) behaviour is used to adapt c to changing network conditions:

- Acknowledgements for segments mean a successful transmission and trigger an additive increase of the congestion window c,

[10] The congestion window is frequently abbreviated as *cwnd* in literature.

- Segment losses are considered as sign of congestion and trigger a multiplicative decrease of the congestion window c.

The state-of-the-art congestion control procedures, which are shortly introduced in the following, are described in [APB09].

2.11.2.1 Increasing the Congestion Window

In order to increase the congestion window c on reception of a SelAck-n-PDU, the following two conditions must have been met:

1. α segments in the SelAck-n-PDU have been *newly* acknowledged, leading to an advance of the send window. That is, new segments have been acknowledged by a CumAck. Previously GapAck'ed segments do *not* count here; they would otherwise lead to a "jump" of α.
2. The congestion window c must have been fully utilised. That is, the maximum allowed number of segments has been outstanding, without causing congestion.

If both conditions are met, c is increased as defined in [APB09, Subsection 3.1]:

$$c = c + \begin{cases} \min\{\alpha, \mathrm{MSS}\} & (c \leq s) \\ \mathrm{MSS} & (c > s \wedge p \geq c) \end{cases}.$$

Here, s denotes the so-called *Slow-Start Threshold*[11] variable s (in segments). For $c \leq s$, the increase phase is denoted as *Slow Start* and leads to an exponential growth of c. Advancing c by the minimum of the acknowledged segments α and one MSS is denoted as *Appropriate Byte Counting* in [All03]. It avoids that a receiver (e.g. a download user) is able to "tune" its incoming data rate by sending multiple small (i.e. $\alpha < \mathrm{MSS}$) acknowledgements to get c increased by a full MSS each time. This would in result lead to a higher (and therefore unfair) bandwidth share for the cheating user.

The second case, i.e. $c > s$, is denoted as *Congestion Avoidance* phase. Here, c is only increased by one MSS when a full window c has been acknowledged. [APB09, Subsection 3.1] recommends to decide whether enough data has been acknowledged to advance c during congestion avoidance by adding another variable p ("partially acknowledged"), which counts the previous acknowledgements. A window advance is possible for $p \geq c$; on advance, p is reset.

2.11.2.2 Decreasing the Congestion Window

For decreasing the congestion window, various variants are possible. In the context of this thesis, only the variant from [APB09, Subsection 3.1] – which makes use of selective acknowledgements (see Subsubsection 2.9.2.2) – is relevant. In case of a fast or timer-based retransmission for a segment j, the congestion window c is decreased as follows:

$$\begin{aligned} s &= \max\left\{\frac{c}{2}, 2 * \mathrm{MSS}\right\}, \\ c &= \begin{cases} s & \text{(Fast Retransmission)} \\ \mathrm{MSS} & \text{(Timer-Based Retransmission)} \end{cases}. \end{aligned}$$

That is, the slow-start threshold s is halved, with a minimum of $2 * \mathrm{MSS}$. A timer-based retransmission of segment j leads to resetting the congestion window c to a single MSS, i.e. the minimum

[11] The slow-start threshold is frequently abbreviated as *ssthresh* in literature.

useful value for an efficient transfer (see also Subsubsection 2.4.1.2). The congestion control therefore goes into slow start mode.

On a fast retransmission of segment j, the congestion window c is set to the slow-start threshold s, i.e. the congestion control goes into congestion avoidance mode. Furthermore, the fast retransmission leads to entering the so-called *Fast Recovery* mode defined in [BAFW03]: the sequence number h of the highest acknowledged segment is remembered (i.e. $h > j$). As long as the fast recovery mode is active, any processing of a newly incoming SelAck-n-PDU must *not* change the congestion window c. The fast recovery mode is left when all segments up to h have finally been CumAck'ed, or if a timer-based retransmission has become necessary.

2.11.3 A Congestion Control Example

A window-based congestion control example for three flows sharing a common bottleneck is depicted in Figure 2.10; the plots are the result of an SCTP simulation based on the model described later in Chapter 5. At time t=0 s, the first flow (see Subfigure 2.10(a)) starts. Initially, the slow-start threshold of Flow #1 is set to the size of the peer receiver window (see Subsection 2.10.1), which is 125,000 bytes in this example. The congestion window of Flow #1 is increased from 4,380 bytes (the initial value of SCTP) using slow start. At around t=2.1 s, a loss results in a fast retransmission. In the following, the flow remains in congestion avoidance, always trying to increase its throughput (as described in Subsubsection 2.11.2.1) until a loss – and therefore a fast retransmission – occurs. The loss event then leads to halving the congestion window (as described in Subsubsection 2.11.2.2) and continuing in congestion avoidance.

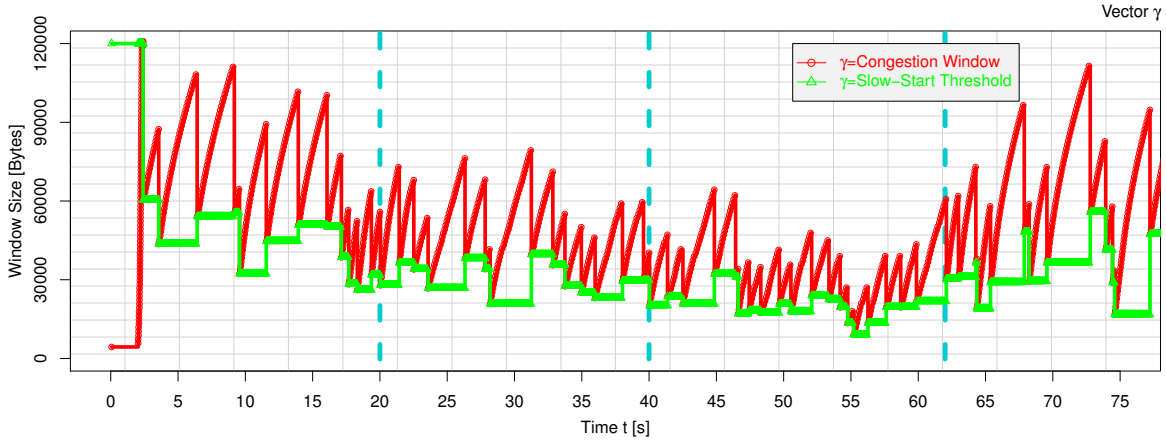
The starts of Flow #2 (at t=20 s; see Subfigure 2.10(b)) and Flow #3 (at t=40 s; see Subfigure 2.10(c)) lead to a reduction of the average congestion window of Flow #1. The result is an equal throughput share for each of the active flows. After the additional flows have been stopped at t=62 s, the congestion window of Flow #1 is able to grow larger again. Then, Flow #1 is able to fully utilise the available bandwidth again.

2.11.4 Dynamic Retransmission Timeout

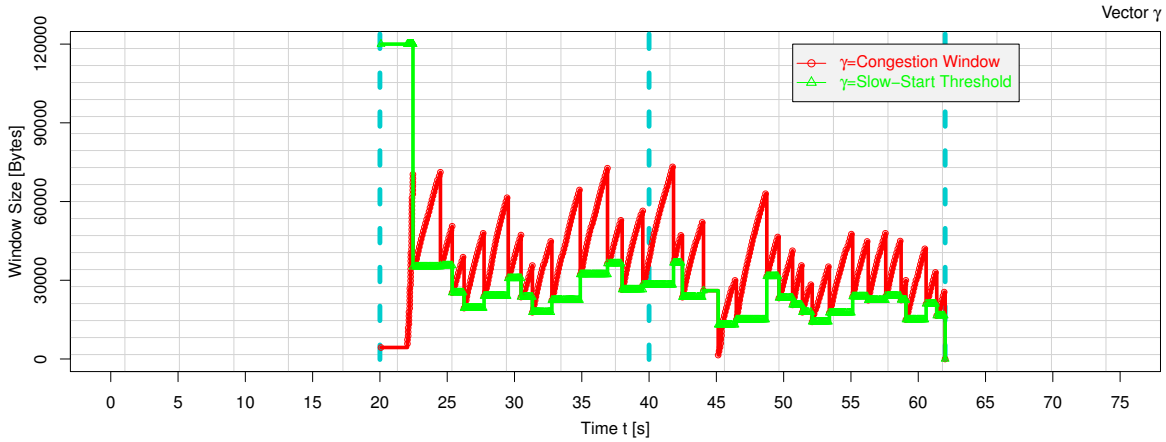
The appropriate configuration of the RTO – i.e. the retransmission timeout as introduced in Subsection 2.9.1 – is crucial for effective congestion control. That is, a too-large RTO leads to an overly long waiting time for acknowledgements, implying a reduced throughput. On the other hand, a too-small RTO leads to retransmission of segments that are still in flight. The latter has significantly contributed to the congestion collapse observed by [Nag84]. Therefore, it is clearly useful to dynamically adapt the RTO, according to the current network state.

While there are various possible ways to set the RTO dynamically, the mechanisms in [PA00] are relevant for the major Internet protocols. Due to their importance in the context of this thesis, they are shortly explained here. The key idea of the described mechanisms is to compute the RTO from a *Smoothed Round Trip Time* (SRTT) as well as the *Round-Trip Time Variance* (RTTVAR), by using RTT measurements (i.e. the time from sending a segment until processing of its acknowledgement, either by CumAck or GapAck). On availability of the first RTT measurement R_0, the variables are initialised:

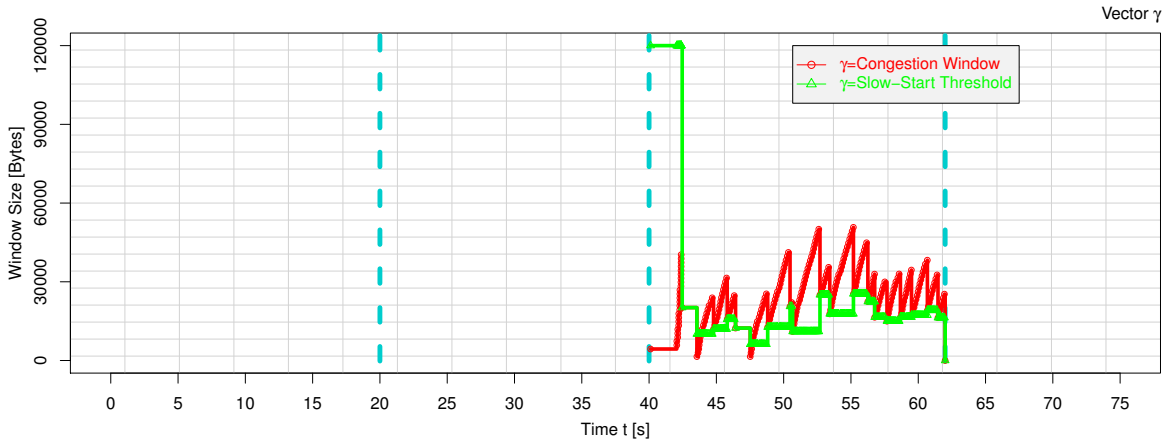
$$\begin{aligned} \mathrm{SRTT}_0 &= R_0, \\ \mathrm{RTTVAR}_0 &= \frac{R_0}{2}, \\ \mathrm{RTO}_0 &= \mathrm{SRTT}_0 + \max\{G, 4 * \mathrm{RTTVAR}_0\} \end{aligned}$$



(a) Flow #1



(b) Flow #2



(c) Flow #3

Figure 2.10: A Three-Flow Congestion Control Example

G denotes the *Clock Granularity*, i.e. the steps of the clock used for the RTT measurements. It depends on the used hardware, of course; a clock counting the time in steps of 10 ms has a setting of $G = 0.010\ s$.

Each following measurement of the RTT R_i results in an update of the variables:

$$\begin{aligned} \mathrm{RTTVAR}_i &= (1-\beta) * \mathrm{RTTVAR}_{i-1} + \beta * |\mathrm{SRTT}_{i-1} - R_i|\,, \\ \mathrm{SRTT}_i &= (1-\alpha) * \mathrm{SRTT}_{i-1} + \alpha * R_i, \\ \mathrm{RTO}_i &= \mathrm{SRTT}_i + \max\{G, 4 * \mathrm{RTTVAR}_i\}. \end{aligned}$$

Here, α and β denote *Smoothing Factors*, which are used to straighten out outliers in the RTT measurements; [PA00, Section 2] recommends[12] the settings of $\alpha = \frac{1}{8}$ and $\beta = \frac{1}{4}$.

In order to accurately measure the RTT, and in result to appropriately configure the RTO, two mechanisms are necessary. These mechanisms have been proposed by [KP91] and are commonly known as *Karn's Algorithm* (see also [PA00, Section 3]):

1. Calculating the RTT of retransmitted segments may lead to ambiguity: it is not known whether the RTT belongs to the initial transmission of a segment, or to a retransmission. Therefore, retransmitted segments are simply ignored in the RTT computation.

2. When the retransmission timer expires, the RTO is doubled ("back off the timer"). This ensures that the RTO increases when there is a sudden delay jump – even if all segments (and therefore acknowledgements for RTT calculation) get lost.

2.11.5 Performance Improvements

Since classic best effort networks have no mechanisms to inform a sender about congestion, their only possibility is to drop PDUs on overload. Therefore, reliable transfer usually assumes that all segment losses are the result of congestion. However, using network access technologies like wireless transmission, data corruption occurs frequently, e.g. due to interferences. This leads to a non-optimal throughput, since each loss – regardless of whether it has been caused by real congestion or data corruption – leads to a reduction of the congestion window.

The approach of *Explicit Congestion Notification* (ECN) allows the network to notify endpoints about congestion by PCI information, instead of just dropping packets. An ECN-capable congestion control may use this information to differentiate between congestion and other kinds of segment losses (e.g. hardware problems or data corruptions), resulting in an improved throughput. However, although ECN has also been defined for the Internet in [RFB01] and provides significant performance improvements as shown by [Kuz05], it is not widely deployed yet. More details on this kind of congestion control improvement can also be found in [Wel05]. Since these mechanisms are not directly relevant in the context of this thesis, a further introduction is omitted here.

2.12 Protocol Standardisation

A certain service may be realised by an almost arbitrary number of completely different protocols. However, in order to ensure interoperability among different applications and devices, it is useful to commonly agree on one protocol (or at least a very small number of protocols). This is denoted as *Standardisation*. A number of international standardisation organisations is existing for this particular

[12] These recommendations are based on an analysis by [JK88].

purpose. In the following, the organisations and protocols that are relevant in the context of this thesis will be briefly introduced.

2.12.1 Wide Area Network Standards

The most important organisation for *Wide Area Network* (WAN) – i.e. large-scale international telecommunication – standards is the *International Telecommunication Union*[13] (ITU), which consists of three sectors:

Radiocommunication Sector (ITU-R) is responsible for allocating radio frequencies and satellite orbits.

Telecommunication Development Sector (ITU-D) takes care of the non-technical development of international telecommunications.

Telecommunication Standardisation Sector (ITU-T) develops standards for telecommunications. It is the former *Comité Consultatif International Téléphonique et Télégraphique* (CCITT).

Relevant in the context of this thesis are some of the Physical Layer/Data Link Layer standards published by the ITU-T. They will be very briefly introduced in the following. For a more detailed introduction, see [Tan96, KR08].

One well-known standard is the *Integrated Services Digital Network* (ISDN), the fully-digital phone network which is the successor of the analogue POTS. ISDN provides a connection-oriented, bidirectional, stream-oriented and unicast communication service with a throughput of 64 Kbit/s per connection. Two connections may be bundled to achieve a throughput of 128 Kbit/s. A detailed introduction to ISDN can be found in [Tan96, Section 2.5].

Furthermore, in conjunction with the ATM Forum (now: Broadband Forum[14]), the ITU-T has been involved in the standardisation of *Asynchronous Transfer Mode* (ATM), a high-speed telecommunications network. It provides a connection-oriented, bidirectional, datagram-oriented and unicast communication service with IntServ support. Furthermore, ATM connections can also be used for transferring data with variable throughputs. Details on ATM can e.g. be found in [RW97]; some details on transferring multimedia traffic over ATM networks are described in [Ste00].

Strongly related to ATM – and as well standardised by the ITU-T – is the *Digital Subscriber Line* (DSL) family of technologies. A detailed overview can be found in [JDS06]. It provides standards for multiplexing data transmissions over analogue telephone lines, in order to connect subscriber networks to *Internet Service Providers* (ISP) for providing Internet access. The ITU-T DSL standards differ in maximum distance, as well as in maximum throughputs from ISP to subscriber (so-called *Downstream*) and from subscriber to ISP (so-called *Upstream*). For a more detailed overview of DSL variants, see [BMR07, All07].

In the context of this thesis, only the variant *Asymmetric Digital Subscriber Line* (ADSL) is relevant: it provides faster downstream and slower upstream throughputs. Due to its focus on high download speed, it is mainly intended for home users. In Germany, it is by far the most popular access technology for Internet connectivity. At the subscriber side, a modem (which is denoted as *ADSL Transceiver Unit – Remote* (ATU-R)) connects to a modem (which is denoted as *ADSL Transceiver Unit – Central Office* (ATU-C)) at a central office of the ISP. The ATU-C is part of a device called

[13] ITU: http://www.itu.int/.
[14] Broadband Forum: http://www.broadband-forum.org/.

Digital Subscriber Line Access Multiplexer (DSLAM); it multiplexes DSL services from many subscribers over an ATM network to a Broadband Remote Access Server (BRAS). The BRAS terminates the subscriber lines and interconnects them to the Internet.

A further notable family of standards published by the ITU-T, in cooperation with the 3rd Generation Partnership Project[15] (3GPP), is *International Mobile Telecommunications 2000* (IMT-2000). It includes the *Universal Mobile Telecommunications System* (UMTS), which is becoming increasingly popular for providing mobile high-speed Internet access to smartphones and other mobile devices.

2.12.2 Local Area Network Standards

Unlike WANs, *Local Area Networks* (LAN) are restricted to a single administrative domain (e.g. a department or a company). Therefore, WAN and LAN standardisation have traditionally been independent. The major organisation for the standardisation of LAN protocols is the *Institute for Electrical and Electronics Engineers*[16] (IEEE), in particular the IEEE 802 working group.

In the context of this thesis, two IEEE 802 Physical Layer/Data Link Layer standards are relevant:

IEEE 802.3 describes *Ethernet*, a standard for communications over copper or fibre wires. It provides a bidirectional, datagram-oriented, connection-less frame transfer using transmission speeds ranging from 10 Mbit/s to currently 100 Gbit/s. The usual MTU is 1,500 bytes, although newer components can be configured for an MTU of 9,000 bytes (in this case, *all* devices in the network must support this MTU). Corruption of frames is detected by using a 32-bit CRC (see Subsubsection 2.5.1.3); corrupted frames are simply ignored. Besides unicast transmission, Ethernet also supports multicast and broadcast communications.

IEEE 802.11 defines *Wireless LAN* (WLAN), a standard for short-range wireless communications. Similar to Ethernet, it provides bidirectional, datagram-oriented, connection-less frame transfer, with support for unicast, multicast and broadcast communications, usually using an MTU of 1,500 bytes. The data frame format is the same as for Ethernet, making data frame exchange on the Data Link Layer possible (so-called *Bridging*). WLAN automatically adapts the transmission speed from 1 Mbit/s to currently 600 Mbit/s, depending on hardware capabilities and interferences.

A detailed introduction to Ethernet, WLAN and the various other IEEE 802 standards can be found in [KR08, Tan96].

2.12.3 Internet Standards

Using a common Network Layer, the *Internet* interconnects LANs over WANs. The most important organisation for the standardisation of Internet protocols (mostly Network Layer and above) is the *Internet Engineering Task Force*[17] (IETF). Since the research and development ideas from this thesis are contributed to the IETF standardisation of SCTP, the IETF – as well as its standardisation process – are briefly described here. A much more detailed overview of the IETF and its standardisation process can be found in [HH06].

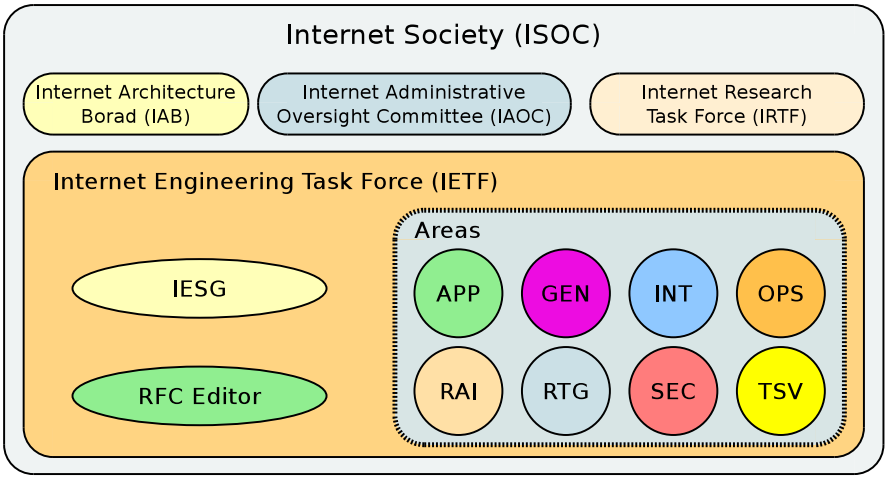


Figure 2.11: An Overview of the Internet Society

2.12.3.1 Overview

The IETF is a part of the *Internet Society*[18] (ISOC), a membership organisation similar to the IEEE. Figure 2.11 depicts the structure of the ISOC. Further parts are:

Internet Architecture Board (IAB) being responsible for the long-range planning and coordination of IETF activities (i.e. the "big picture" of the Internet);

Internet Administrative Oversight Committee (IAOC) taking care of the administrative tasks of the ISOC (particularly, planning and organisation of IETF meetings and development of tools for the work of IETF working groups); as well as

Internet Research Task Force (IRTF) being concerned with longer-term research issues related to the Internet.

The IETF – in contrast to the IRTF – focusses on the shorter-term issues of engineering and standardisation. It is organised into currently eight areas, which are shown in Table 2.4. Every area consists of a set of *Working Groups* (WG), each dealing with certain topics defined in the milestones of the WG. Of interest – in the context of this thesis – is only the *Transport Services Area* (TSV), containing the following relevant WGs:

Transport Services Working Group (TSVWG) is responsible for SCTP, but also for TCP extensions, as well as for DiffServ and IntServ.

Signalling Transport (SIGTRAN) has standardised signalling transport protocols for transporting telephone signalling (e.g. used for ISDN) over the Internet.

Reliable Server Pooling (RSerPool) is responsible for server pool and session management.

[15] 3GPP: http://www.3gpp.org/.
[16] IEEE: http://www.ieee.org/.
[17] IETF: http://www.ietf.org/.
[18] ISOC: http://www.isoc.org/.

Area	Description
Applications (APP)	Protocols seen by user programs (e.g. email and the web)
General (GEN)	Catch-all for WGs that do not fit in other areas
Internet (INT)	Different ways of moving IP packets and DNS information
Operations and Management (OPS)	Operational aspects, network monitoring, configuration
Real-time Apps. and Infrastr. (RAI)	Delay-sensitive interpersonal communications
Routing (RTG)	Getting packets to their destinations
Security (SEC)	Authentication and privacy
Transport Services (TSV)	Special services for special packets

Table 2.4: The Areas of the IETF

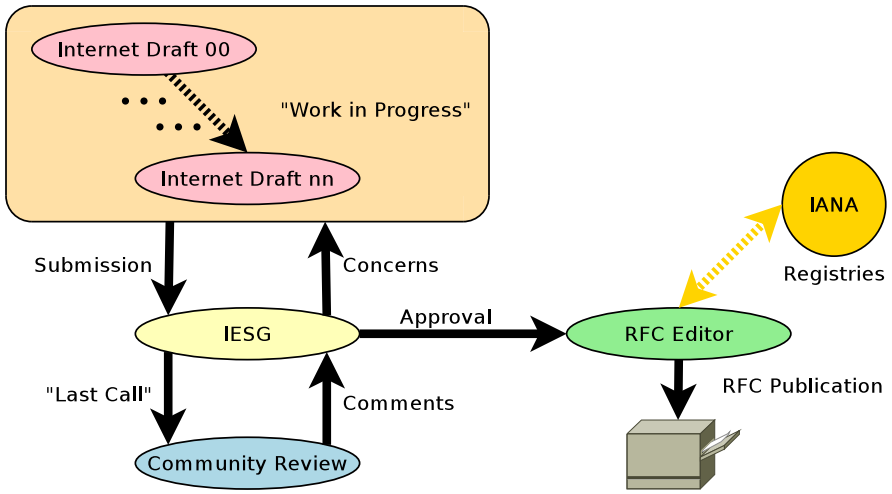


Figure 2.12: The IETF Document Lifecycle

A starting point for a more detailed overview on the other seven areas and their various WGs is provided by [Bra05a]. It also gives a more detailed introduction to the structure of the ISOC.

RFC Editor[19] and *Internet Engineering Steering Group*[20] (IESG) are important entities in the standardisation process; their roles will be described below.

2.12.3.2 Standardisation Process

The IETF standardisation process – described in detail in [Bra96] – is illustrated in Figure 2.12: proposals for standardisation are submitted in form of an *Internet Draft* (I-D), either as so-called *Individual Submission* (document name prefix is then "draft-*[submitter name]*", e.g. "draft-tuexen-rserpool-policies") or as *Working Group Draft* (document name prefix is then "draft-ietf-*[WG name]*"; e.g. "draft-ietf-rserpool-policies"). The usual case is that a draft starts as an individual submission. After that, it is discussed on IETF mailing lists and/or at IETF meetings, and iteratively improved. When discussions have reached a rough consensus, the draft may be adopted by the corresponding

[19] RFC Editor: http://www.rfc-editor.org/.
[20] IESG: http://www.ietf.org/iesg/.

WG as working group draft, and furthermore iteratively improved again. Each improvement cycle results in re-submitting the updated draft under the next version number. The version number is appended to the document name in form of a two-digit decimal number; the initial version is 00, e.g. "draft-ietf-rserpool-policies-00". That is, Internet Drafts may be considered as "work in progress".

When there is finally a rough consensus that a draft is "ready", it is submitted to the IESG for approval. The IESG performs a "Last Call", i.e. a final request for comments, on the main IETF mailing list. Everybody interested in the document is invited to review it, i.e. a public and open community review of the document is performed. For protocol documents, the IETF standardisation process strongly relies on "running code", i.e. it is required that at least a "proof of concept" implementation, or – even better – multiple, independently-developed and interoperable implementations, are available at this point. In case of raised concerns or problems, the IESG will return the draft for further improvement, and the update procedure repeats. When finally the "Last Call" reaches a rough consensus, the draft is approved by the IESG and forwarded to the RFC Editor.

The RFC Editor makes final editorial changes (e.g. correcting typos and grammar, resolving reference dependencies, etc.). Also, if necessary, the RFC Editor interacts with the *Internet Assigned Numbers Authority*[21] (IANA) for number registrations. The IANA manages various registries (e.g. protocol numbers, standard TCP ports, etc.); a draft may need such registrations, or even create a completely new registry. IANA considerations are written in the corresponding section of a draft; [NA08] describes these procedures in detail.

After the changes of the RFC Editor have been approved by the author(s), the document is published as a new *Request for Comments* (RFC), with a unique RFC number (e.g. RFC 5351 for the draft example above). RFCs represent the final standards documents of the IETF. Due to the community review process, citations of RFCs may be considered equivalent to scholarly publications, as discussed in [CP10].

Some further important information on Intellectual Property Rights (IPR) in RFCs can be found in [Bra05b]. IPR in this context mainly denotes software patents. Actually, it is possible for an RFC to contain patented technologies, as long as they can be implemented under "reasonable and non-discriminatory terms". However, the IESG intentionally does not make any explicit determination of these terms, as described in [Bra05b, Subsection 4.1]. In practise, this means that patented technologies will only become part of an RFC if no suitable, patent-free alternative is available. As described by [Dre07, Subsection 3.10.5], an extension of the Internet-16 checksum (see Subsubsection 2.5.1.1) to more than 16 bits *might* be covered by a Motorola patent. Therefore, the existing Internet-16 algorithm instead of a 32-bit version has been used in the RFC 5353 (i.e. the ENRP protocol defined in [XSS+08]; to be introduced in Section A.2).

2.13 Internet Protocols

In the following, the IETF protocols that are relevant for this thesis are briefly introduced. The focus of this introduction is to bring the protocols into the context of the OSI reference model and the explained basic protocol mechanisms. Detailed introductions to the internal protocol procedures are omitted here, since this would greatly exceed the scope of this overview; they can be found in the referenced RFCs.

[21] IANA: http://www.iana.org/.

2.13.1 Physical Layer and Data Link Layer

An important Data Link Layer protocol standardised by the IETF in [Sim94] as RFC 1661 is the *Point-to-Point Protocol* (PPP). It provides an adaptation service for a unicast, bidirectional, connection-oriented, datagram-oriented communication over a stream-oriented channel (see Subsection 2.3.3), using a CRC checksum of 16 or 32 bits for corruption detection (see Subsubsection 2.5.1.3) as well as a sliding-window-based reliable transfer (see Subsection 2.9.2) and flow control (see Subsection 2.10.1). It is in widespread use for Internet communication over modem and ISDN channels. In the context of this thesis, the *PPP over Ethernet* (PPPoE) variant – defined in [MLE+99] as RFC 2516 – is relevant: it is frequently used for ADSL communication, where Internet communications between BRAS and subscriber-side router are transported via Ethernet frames. Since PPPoE uses a header of 8 bytes, it leads to a Layer 2-MTU of 1,492 bytes (if transported over Ethernet with its standard MTU of 1,500 bytes).

The IETF has also standardised some special-purpose[22], high-delay Physical Layer protocols, notably Avian Carrier (in [Wai90] as RFC 1149), Semaphore Flags (in [HBZ07] as RFC 4824) and IPv6 over Social Networks (in [Vyn09] as RFC 5514). These standards include their own Data Link Layer, providing a unicast, bidirectional, connection-oriented, datagram-oriented communication service.

2.13.2 Network Layer

Clearly, the relevant Network Layer protocols of the IETF are the *Internet Protocol, version 4* (IPv4) – defined in [Pos81b] as RFC 791 – as well as its successor, the *Internet Protocol, version 6* (IPv6) – defined in [DH98] as RFC 2460. Both protocols are so-called *Routed Protocols*, i.e. they take care of forwarding packets among endpoints. Therefore, a key functionality provided by IPv4 as well as IPv6 is hierarchical addressing; IPv4 uses a 32-bit address space, while IPv6 uses a 128-bit one. In the following, both protocol versions are commonly denoted as Internet Protocol (IP), unless referring to a specific version.

IP is connection-less, i.e. each packet may take its individual trail through the network. Routing is performed on a hop-per-hop basis. That is, each node (i.e. router or endpoint) uses a so-called *Routing Table* to look up the next hop to forward a packet to its given destination. The setup of routing tables is out of the scope of IP; they may be configured statically, or – in large networks – dynamically by a *Routing Protocol*.

IP provides bidirectional, datagram-oriented communication with support for unicast, multicast as well as broadcast. Only unreliable transfer and unordered delivery, with neither flow nor congestion control, are supported. Furthermore, only IPv4 provides corruption detection using the Internet-16 checksum algorithm (see Subsubsection 2.5.1.1) for its *header*, but not for its payload. IPv6 omits corruption detection altogether, in order to improve forwarding efficiency. Any more advanced feature – if needed for a certain service built on top of IP – has to be provided by a higher layer.

IP supports segmentation and reassembly of large 3-SDUs over a small Layer 2-MTU:

- IPv4 defines no Layer 2-MTU, but it requires the receiver side to accept or reassemble packets of at least 576 bytes. The maximum supported 3-SDU size is 65,535 bytes (i.e. $2^{16} - 1$ bytes). A 16-bit segment sequence number and a byte offset are used for reassembly (see Section 2.7). Each IPv4 node (i.e. endpoints as well as routers) may perform segmentation.

- IPv6 requires a Layer 2-MTU of at least 1,280 bytes. For a smaller low-layer MTU, an adaptation layer like PPP (see Subsection 2.13.1) may be used below IPv6. In contrast to IPv4, only

[22] The reader is suggested to take note of the publication date of these IETF standards documents.

the sender endpoint may perform segmentation. Also, IPv6 applies a 32-bit segment sequence number and a byte offset for reassembly (see Section 2.7). Both constraints allow a more efficient packet forwarding. Similar to IPv4, the maximum 3-SDU size is 65,535 bytes. However, by applying the *IPv6 Jumbogram* extension defined in [BDH99] as RFC 2675, the limit can be increased to $2^{32} - 1$.

In conjunction with IP, which is used for the payload data transfer, the *Internet Control Message Protocol* (ICMP), with its variants ICMPv4 for IPv4 (defined in [Pos81a] as RFC 792) and ICMPv6 for IPv6 (defined in [CDG06] as RFC 4443), is used for connectivity testing and error reporting. ICMP is built on top of IP, it also serves for certain kinds of error handling for Transport Layer protocols. Therefore, its layer in the OSI reference model can be seen as 3.5, i.e. between Network and Transport Layer. The particularly relevant feature of ICMP in the context of this thesis is the so-called *Path MTU Discovery*, which is used to find out the MTU of a certain path (i.e. with respect to the trails the packets actually take; see also Subsection 2.1.3 and Subsection 2.1.5). Knowledge of the path MTU is crucial for IPv6, since the sender must segment its packets accordingly. Furthermore, higher-layer protocols with built-in segmentation and reassembly features may improve transfer efficiency by using this information. The actual procedure for path MTU discovery is defined in [MH07] as RFC 4821, the handling of common problems – in particular badly configured firewalls dropping ICMP packets – is described in [Lah00] as RFC 2923.

A detailed introduction to IP can be found in [KR08, Chapter 4], an introduction to Network Layer protocols in general is provided by [Tan96, Section 5.5].

2.13.3 Transport Layer

An important task of the Transport Layer is to identify applications on endpoints. For this purpose, the four IETF Transport Layer protocols being relevant in the context of this thesis use so-called *Ports*, which are endpoint-unique 16-bit application addresses.

The *User Datagram Protocol* (UDP) – defined in [Pos80] as RFC 768 – is the simplest of the four Transport Layer protocols. Like IP, it provides connection-less, bidirectional, datagram-oriented communication with support for unicast, multicast as well as broadcast. Also, only unreliable transfer and unordered delivery, with neither flow nor congestion control are supported. UDP only adds the mentioned application identification by ports, as well as data corruption detection – covering the whole UDP-PDU – by applying the Internet-16 checksum algorithm (see Subsubsection 2.5.1.1).

Like UDP, the *Datagram Congestion Control Protocol* (DCCP) – which is defined in [KHF06] as RFC 4340 – provides bidirectional, datagram-oriented communication with unreliable transfer and unordered delivery. However, DCCP only supports unicast communication, and it works connection-oriented. Its key feature is the support of window-based congestion control (see Subsection 2.11.2), including support for ECN (see Subsection 2.11.5). The actual congestion control algorithm is negotiated between the endpoints. Corruption detection is provided by an Internet-16 checksum for the DCCP header as well as the DCCP payload (completely or in part). Furthermore, it is possible to optionally use a CRC-32C checksum (see Subsubsection 2.5.1.3) for the payload.

Reliable transfer and ordered delivery with window-based flow and congestion control for bidirectional, connection-oriented and unicast communication is provided by the *Transmission Control Protocol* (TCP), which is defined in [Pos81c] as RFC 793. Originally, TCP has only provided support for CumAcks (see Subsection 2.9.2); the selective acknowledgements extension defined in [MMFR96] – which is now used by all major implementations – also adds GapAcks (see Subsubsection 2.9.2.2). State-of-the-art TCP implementations – as described in [APB09] as RFC 5681 – furthermore support

selective repeat (see Subsubsection 2.9.2.2), delayed acknowledgement (see Subsubsection 2.9.3.1), fast retransmission (see Subsubsection 2.9.3.4) and ECN (see Subsection 2.11.5). Unlike UDP and DCCP, TCP provides stream-oriented communication. The size of a TCP segment is one byte; bundling is used to fill the MSS. Like UDP and DCCP, TCP uses the Internet-16 checksum algorithm (see Subsubsection 2.5.1.1) to detect data corruption. Corrupted TCP-PDUs are simply dropped and retransmitted (see Subsubsection 2.9.3.5).

Like TCP, the *Stream Control Transmission Protocol* (SCTP) – which is defined in [Ste07] as RFC 4960 – provides reliable transfer with window-based flow and congestion control for connection-oriented, bidirectional, datagram-oriented and unicast communication. SCTP will be introduced in detail in Chapter 3.

A more detailed introduction to the Transport Layer protocols UDP and TCP is e.g. provided by [Tan96, Section 6.4] as well as by [KR08, Chapter 3].

2.13.4 Session Layer, Presentation Layer and Application Layer

The first and currently only Session Layer protocol defined by the IETF is the *Aggregate Server Access Protocol* (ASAP) – defined in [SXST08a] as RFC 5352 – which is part of the Reliable Server Pooling framework. It will be introduced shortly in Appendix A.

A well-known IETF Presentation Layer standard is *Portable Network Graphics* (PNG), a lossless image format defined in [Bou97] as RFC 2083. Furthermore, the IETF has standardised the *Hypertext Markup Language* (HTML), version 2.0, in [BLC95] as RFC 1866. It is used as markup language for web pages. However, all later versions are maintained[23] by the *World Wide Web Consortium* (W3C). A short introduction to HTML is provided by [Ste00, Section 20.5].

Notable IETF Application Layer protocols in the context of this thesis are:

HyperText Transfer Protocol (HTTP) – defined in [FGM+99] as RFC 2616 – to provide file transfer, on top of TCP;

Trivial File Transfer Protocol (TFTP) – defined in [Sol92] as RFC 1350 – to provide a simple file transfer on top of UDP (realising stop and wait – as explained in Subsection 2.9.1 – for ordered delivery as well as flow and congestion control);

Network Time Protocol (NTP) – defined in [MMBK10] as RFC 5905 – to accurately synchronise clocks among endpoints, on top of UDP;

File Transfer Protocol (FTP) – defined in [PR85] as RFC 959 – to provide file management, on top of TCP;

Secure Shell (SSH) Protocol – defined in [YL06] as RFC 4254 – for a secure, interactive remote shell and connection tunnelling, usually on top of TCP; as well as

Extensible Messaging and Presence Protocol (XMPP) – a framework[24] for instant messaging on top of TCP – defined in [SA11b, SA11c, SA11a] as RFC 6120, RFC 6121 and RFC 6122.

Furthermore, protocols for dynamically adapting routing tables – denoted as *Routing Protocols* – also belong to the Application Layer. Important protocols are the *Routing Information Protocol* (RIP) with its variants RIPv2 for IPv4 (defined in [Mal98] as RFC 2453) and RIPng for IPv6 (defined

[23] The current specification of HTML can be found in [Hic11, AM10].

[24] XMPP is frequently denoted as "Jabber", which has been the initial name for this framework.

in [MM97] as RFC 2080), the *Open Shortest Path First* (OSPF) with its variants OSPFv2 for IPv4 (defined in [Moy98] as RFC 2328) and OSPFv3 for IPv6 (defined in [CFM99] as RFC 2740), as well as the *Border Gateway Protocol* (BGP) defined in [RLH06] as RFC 4271 (for both, IPv4 and IPv6). An introduction to the basics of routing protocols and to the protocols themselves can be found in [Tan96, KR08].

2.14 Summary

This chapter has introduced the protocol-independent terminology and mechanisms which are used throughout this thesis. In particular, common basic protocol mechanisms – providing segmentation and reassembly, ordered delivery, reliable transfer, flow control as well as congestion control – have been introduced. Finally, an overview of protocol standardisation, including the standard protocols being relevant in the context of this thesis, has been provided.

Chapter 3

The Stream Control Transmission Protocol

This chapter introduces the SCTP protocol. This introduction first covers the core SCTP protocol itself, including its features, the basic communication procedures as well as the actually used protocol mechanisms. Furthermore, important SCTP protocol extensions are described. This chapter closes with a short overview of available SCTP implementations and important application scenarios.

3.1 Introduction

SCTP is a general-purpose, unicast, bidirectional, connection-oriented and datagram-oriented Transport Layer protocol. It has initially been defined by the IETF SIGTRAN WG, for the purpose of transporting telephone signalling traffic over the Internet (to be explained in Subsection 3.14.1). In October 2001, it has been published in [SXM+00] as RFC 2960. Later, the IETF TSVWG WG has continued the SCTP development; a revised version – based on experiences with implementation and deployment in productive environments – has been published in September 2007 in [Ste07] as RFC 4960. A more user-oriented introduction to SCTP can be found in [SX01]; a summary of the recent IETF standardisation activities on SCTP – which are still ongoing in form of protocol extensions for special use cases – is provided by [DRS+11].

3.2 Packets and Chunks

A connection in the terminology of SCTP is denoted as an *Association*. Like TCP, UDP and DCCP, SCTP also uses the concept of ports to add application addressing (see Subsection 2.13.3). The structure of the SCTP packets (i.e. SCTP-PDUs being transferred as part of an association) is presented in Figure 3.1. This structure is the same for all SCTP packets and contains a fixed 12-bytes header. This so-called *Common Header* contains the source and destination port, defining the source application on the sender side and the destination application of the receiver side.

Furthermore, a 32-bit checksum (see Subsection 2.5.1) is used to detect corruption within the whole SCTP packet. In the initial definition of SCTP in [SXM+00] (RFC 2960), the Adler-32 checksum algorithm had been used (see Subsubsection 2.5.1.2). However, later, the weaknesses of this algorithm had shown to be problematic in certain application scenarios using small packets. The checksum algorithm had therefore been replaced by CRC-32C (see Subsubsection 2.5.1.3), using an

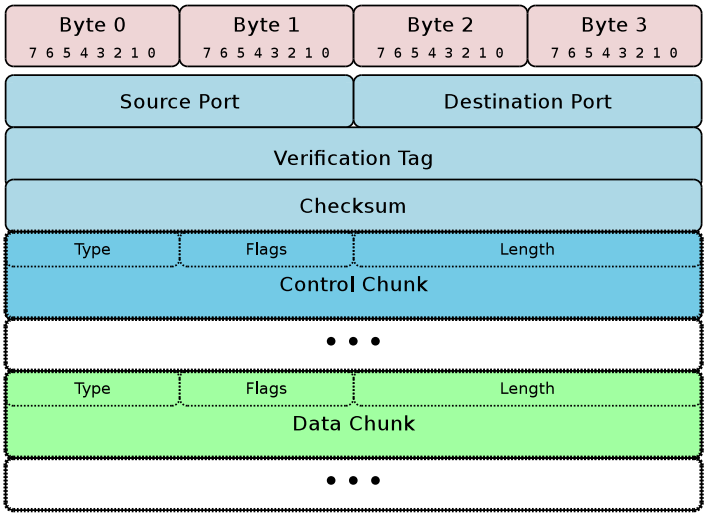


Figure 3.1: The Structure of an SCTP Packet

update to the initial specification in [SSO02] as RFC 3309. Clearly, the current specification in [Ste07] as RFC 4960 also uses CRC-32C. An interesting fact of this change is that instead of making the checksum algorithm configurable, and providing interoperability between old and new implementations, the algorithm had just been replaced. In order to make this replacement possible – which in fact invalidates a part of the original specification – all implementers of SCTP had agreed to make this change.

The last part of the common header is the *Verification Tag*. The verification tag is a random, 32-bit number. It is negotiated during association setup – to be explained in Section 3.3 – for each transport direction. During the whole lifetime of an association, a verification tag remains fixed and must be supplied in each SCTP packet. This mechanism is applied to prevent a blind *Denial of Service* attack: if port numbers are fixed, it would be quite easy for an attacker to insert its own packets into the packet flow of an association, and e.g. cause associations breaks (i.e. service interruptions) or even hijack an association. By applying the verification tag, an attacker not only has to guess the right port numbers but also the correct 32-bit verification tag (i.e. one number in 2^{32} possibilities).

Next to the common header, an SCTP packet includes one or more *Chunks*, which may include user data (*DATA chunk*) or control information (*Control Chunk*). Each chunk has a variable length and consists of a chunk type, some type-specific flags, the length of the chunk, its actual data and possibly a padding which ensures that the total length of the chunk is a multiple of four bytes. Since the chunk format is generic, a receiver entity is able to parse a received packet, even if it does not support some of the contained chunk types. The SCTP specification also defines how to handle unknown types (silently skip the chunk, skip the chunk but report skipping to the sender instance, silently skip the whole packet, skip the whole packet but report an error to the sender instance; see also [Ste07, Subsection 3.2]). This allows for easily adding protocol extensions (to be explained in Section 3.11) by defining new control chunk types.

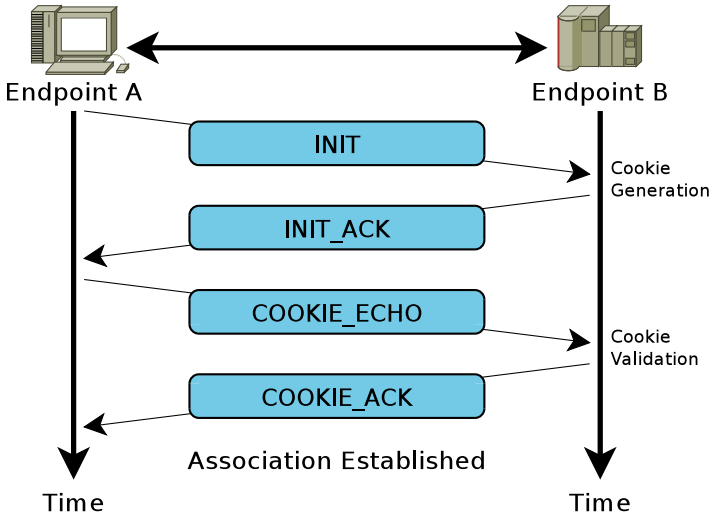


Figure 3.2: The Chunk Sequence of the SCTP Association Setup

3.3 Association Establishment

An association establishment is performed by a so-called *4-Way Handshake*, as shown in Figure 3.2. The initiating side (here: Endpoint A) sends a packet containing an *INIT Chunk*, the peer side (here: Endpoint B) responds with a packet containing an *INIT_ACK Chunk*. It is important to note that Endpoint B does not store any state information yet. Instead, all information necessary for association setup is stored in a so-called *Cookie*, which is a part of the INIT_ACK chunk. Endpoint A must send a copy of this cookie as part of a *COOKIE_ECHO Chunk* back to Endpoint B. A signature in the cookie, with a key only known to Endpoint B, ensures integrity and authenticity. Using the information from the cookie, Endpoint B actually establishes the association and confirms the completion of the setup procedure using a *COOKIE_ACK Chunk*.

The 4-way handshake makes SCTP resilient against denial of service attacks using connection initiation flooding, in contrast to *SYN Flooding* for the *3-Way-Handshake* of TCP (see also [Edd07]): since no connection resources are actually reserved upon reception of an INIT chunk, an attacker cannot exhaust them. To actually set up an association, Endpoint A must be reachable, in order to receive the cookie in an INIT_ACK chunk and send it back in a COOKIE_ECHO chunk. A TCP server – on the other hand – allocates resources on reception of each SYN request (i.e. the first packet of a 3-way handshake; see [Pos81c]), without having any knowledge about the existence of the potential client. Depending on the TCP implementation, resources remain reserved for several minutes, preventing legitimate users of the server to connect to it due to a lack of resources.

The INIT and INIT_ACK chunks are also used to negotiate the verification tag for each transport direction. Furthermore, the configuration of various other protocol options – like the support of certain protocol extensions (to be described in Section 3.11) – is possible.

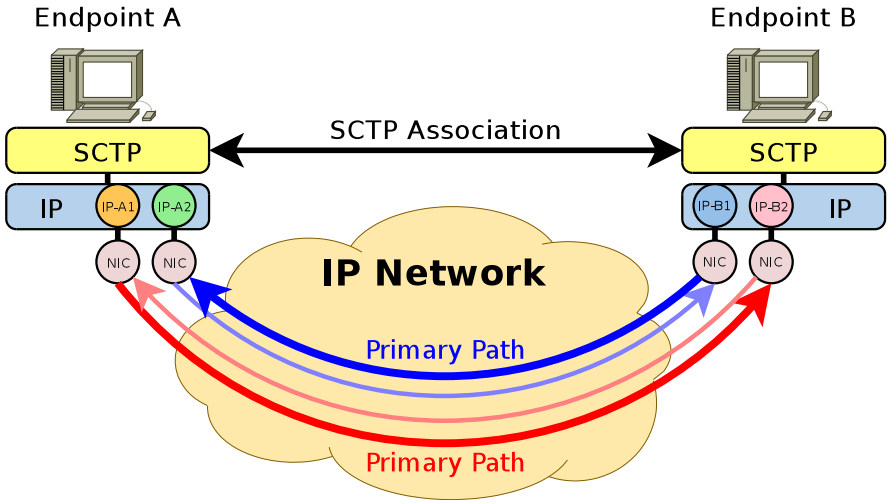


Figure 3.3: The Multi-Homing Feature of SCTP

3.4 Multi-Homing

3.4.1 Principle

Unlike TCP (see [Pos81c]) or DCCP (see [KHF06]), each SCTP endpoint may use multiple Network Layer addresses for the packet transfer of an association. Therefore, during association setup, each SCTP endpoint provides a list of its Network Layer addresses to be used for the association in the INIT and INIT_ACK chunks. In particular, an endpoint may mix IPv4 and IPv6 addresses, i.e. both underlying Network Layer protocols may be used simultaneously. Each Network Layer address of a peer endpoint defines a unidirectional *Path* to that endpoint (this definition is analogue to the formal definition of a path in Subsection 2.1.5). The existence of multiple endpoint addresses is denoted as *Multi-Homing*. Figure 3.3 illustrates the principle: in each direction, *one* path is selected as so-called *Primary Path*. This selected path is used for the actual data transmission. The other paths are used as backup; they are only used for retransmissions (to be explained in Section 3.7) or if a primary path becomes unavailable. Note, that standard SCTP – as defined in [Ste07] – does *not* perform load sharing among its paths; the protocol extension for this feature will be explained in Chapter 4.

3.4.2 Formal Definition

Using the terminology definitions from Section 2.1, it is possible to exactly define an association in a formal way:

Definition 3.4.1. Let $\Gamma = (L, N, C, c)$ be a network. An association $\kappa = (a, b, P_{a\to b}, P_{b\to a})$ between node a and node b in network Γ is defined as follows:

- $a \in N$ – local node,
- $b \in N$ – peer node,
- $P_{a\to b} \subseteq \overline{P}_\Gamma(a, b)$ – paths from node a to node b,
- $P_{b\to a} \subseteq \overline{P}_\Gamma(b, a)$ – paths from node b to node a.

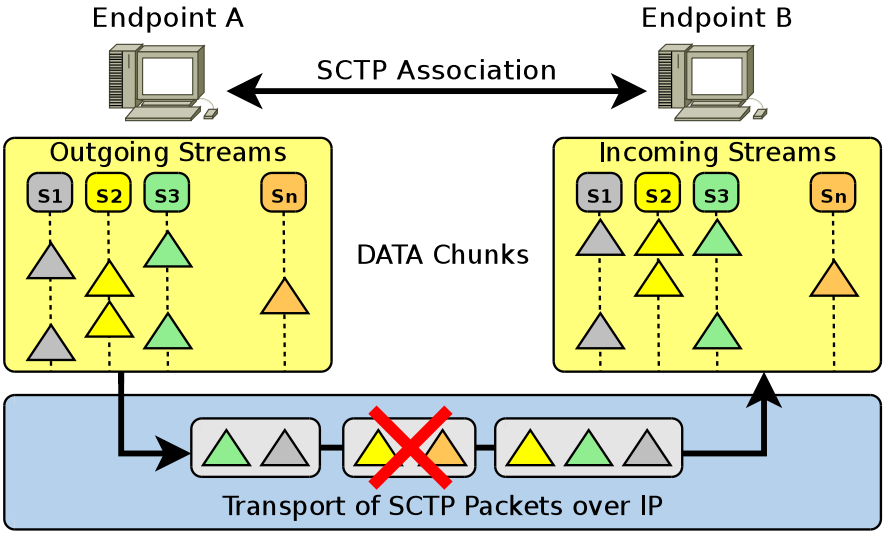


Figure 3.4: The Concept of SCTP Multi-Streaming

The following conditions apply:

- $|P_{a\to b}| \geq 1$ ("connectivity from node a to node b"),
- $|P_{b\to a}| \geq 1$ ("connectivity from node b to node a").

That is, an association between two nodes a and b consists of *at least* one path in each direction. Therefore, $|P_{a\to b}| + |P_{b\to a}| \geq 2$. In order to further emphasise the number of paths, two more terms are defined:

Definition 3.4.2. Let $\kappa = (a, b, P_{a\to b}, P_{b\to a})$ be an association in network Γ.

- κ is denoted as a *Single-Homed* association $\Leftrightarrow |P_{a\to b}| + |P_{b\to a}| = 2$.
- κ is denoted as a *Multi-Homed* association $\Leftrightarrow |P_{a\to b}| + |P_{b\to a}| > 2$.

3.4.3 Path Monitoring

In order to ensure that a path of an association is actually working, so-called *Path Monitoring* is applied. That is, in a configurable interval, each SCTP endpoint sends a *HEARTBEAT Chunk* over a path. Upon reception, the peer side answers a HEARTBEAT chunk with a *HEARTBEAT_ACK Chunk*. When the sender side receives this answer, the path is known to work.

3.5 Multi-Streaming

A further important feature of SCTP is the so-called *Multi-Streaming*. SCTP supports the multiplexing of up to 65,535 (i.e. $2^{16} - 1$) unidirectional, independent data flows – denoted as *Streams* – for each transfer direction over a single association. The number of these streams is negotiated at association setup. Since streams are independent, SCTP does not need to take care of sequential data delivery *across* streams. This avoids the problem of *Head-of-Line Blocking*: if data of a stream is

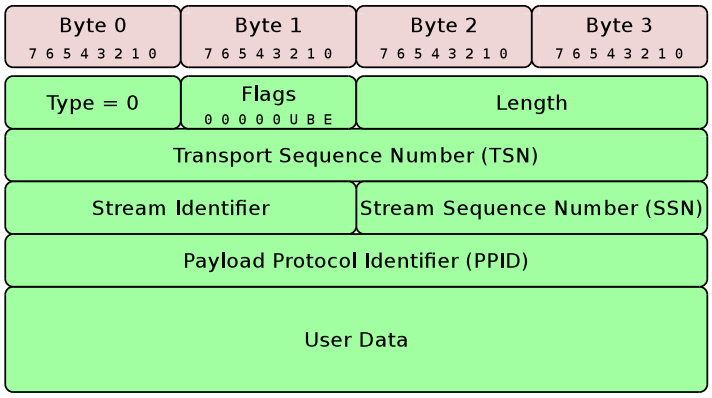


Figure 3.5: The Structure of a DATA chunk

delayed (e.g. if it has to be retransmitted), a coexisting stream does not have to wait for this data. An example is provided in Figure 3.4: Endpoint A uses n streams to Endpoint B. If the packet containing DATA chunks of streams #2 and #n gets lost, only stream #2 and stream #n would have to wait for a retransmission before an ordered delivery of subsequent DATA chunks within these streams. There is no need to delay delivery within the other streams.

3.6 Segmentation and Reassembly

SCTP provides segmentation and reassembly by segmenting the datagrams – which are denoted as *Messages* in the terminology of SCTP – into DATA chunks. The structure of such a DATA chunk is depicted in Figure 3.5.

A 32-bit sequence number (see Section 2.6) – called *Transport Sequence Number* (TSN) – is used to uniquely enumerate the DATA chunks. The initial TSNs for each transport direction are randomly selected and negotiated during association setup. When a message gets segmented, consecutive TSNs are allocated to the resulting DATA chunks. That is, a DATA chunk constitutes a segment of SCTP, in contrast to TCP using a byte as segment. This property becomes important for congestion control, as will be explained in Section 3.8. The first DATA chunk of a message is marked by a set "Begin of Message" bit (B-bit) within the *Flags* field; the last DATA chunk is marked by the "End of Message" bit (E-bit). Since the TSNs of a message are consecutive, message framing can be preserved during reassembly.

SCTP supports multi-homing, and the DATA chunks of a message may possibly take different paths (e.g. due to a retransmission). Therefore, segmentation – as introduced in Section 2.7 – is performed by using the so-called *Smallest Path MTU*. It denotes the smallest known MTU of any of the paths belonging to the association, found out by path MTU discovery as described in Subsection 2.13.2. It should therefore be a safe setting. However, for the – probably extremely rare – case of an abrupt decrease of an MTU, SCTP relies on the underlying Network Layer protocol to segment already-created DATA chunks appropriately. This avoids the introduction of high complexity to re-allocate TSNs in order to handle this event.

The number of the stream a DATA chunk belongs to is given in the *Stream Identifier* field; the *Stream Sequence Number* (SSN) is a 16-bit sequence number that provides the sequence of a message for ordered delivery within its stream (starting from zero). SCTP provides ordered as well as

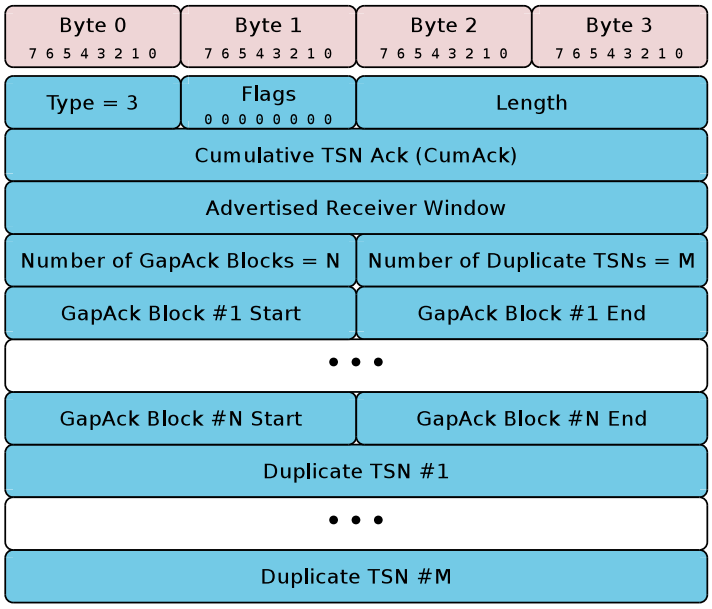


Figure 3.6: The Structure of a SACK Chunk

unordered delivery per message. The "Unordered" bit (U-bit) within the Flags field denotes whether the message of the DATA chunk must be delivered in-sequence (U=0, i.e. ordered delivery) or may be delivered out-of-sequence (U=1, i.e. unordered delivery).

The *Payload Protocol Identifier* (PPID) provides the number of the upper-layer protocol transported via SCTP. That is, within a single stream, multiple different upper-layer protocols may be multiplexed; the PPID can be used to differentiate among these protocols. The IANA runs a registry for SCTP PPIDs[1] (see also Subsubsection 2.12.3.2). Last entry of the DATA chunk is the variable-length *User Data* field. It contains the actual segment payload.

3.7 Reliable Transfer

Besides the segmentation and reassembly, the TSNs are also used for providing reliable transfer using a sliding-window-based mechanism with selective repeat. The necessary acknowledgement mechanism is realised by a *SACK Chunk*, containing a selective acknowledgement. Its structure is depicted in Figure 3.6.

The *Cumulative Acknowledgement* field contains the CumAck for the highest in-sequence TSN received. Further TSN GapAck ranges may be provided as variable-length list, defining the first and the last GapAck'ed TSN of a block. The number of these blocks is given in the *Number of GapAck Blocks* field. Furthermore, the SACK may contain the TSNs of DATA chunks which have been received multiple times (i.e. being duplicates), as variable-length list of *Duplicate TSNs*. The number of these duplicate TSN entries is given in the *Number of Duplicate TSNs* field.

SCTP furthermore applies the protocol mechanisms of bundling (see Subsubsection 2.9.3.3), piggybacking (see Subsubsection 2.9.3.2), delayed acknowledgement (see Subsubsection 2.9.3.1) and fast retransmission (see Subsubsection 2.9.3.4) with fast recovery (see Subsubsection 2.11.2.2). Since

[1] IANA SCTP Parameters Registry: http://www.iana.org/assignments/sctp-parameters.

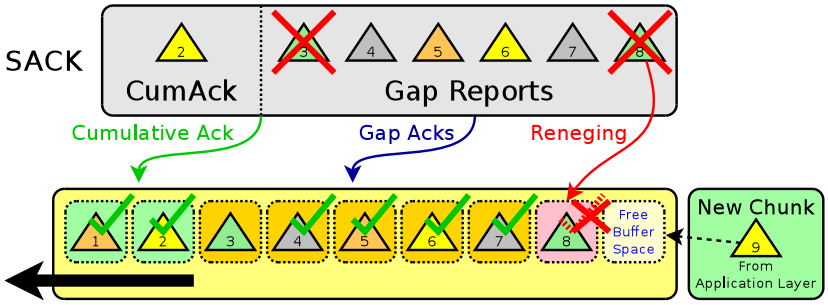


Figure 3.7: A Selective Acknowledgement Example

multiple chunks may be placed in a single packet, SCTP delays the transfer of a new packet by default for up to 200 ms, in order to try utilising a complete MTU. Furthermore, a SACK chunk is by default generated only for every second received *packet* containing a DATA chunk (i.e. the number of DATA chunks in these packets is not relevant), being received within 200 ms of the arrival of any unacknowledged DATA chunk (see also [Ste07, Subsection 6.2]). These mechanisms reduce the acknowledgement overhead. But when receiving a DATA chunk having an out-of-sequence TSN, a SACK chunk is sent immediately. This is useful to quickly tell a sender about missing DATA chunks. Missing DATA chunks are handled in two different ways:

- Once a DATA chunk has been reported as missing for three times (by default), it is retransmitted immediately on the same path as fast retransmission (as defined in [Ste07, Subsection 7.2.4]; see also Subsubsection 2.9.3.4).
- Any further retransmissions are timer-based retransmissions (see also Subsubsection 2.9.3.4). If possible, they should use alternative paths (because the original path seems to have problems, since two previous transmissions have been lost already).

A SACK chunk is also used for flow control (see also Section 2.10): the *Advertised Receiver Window* field contains the number of *payload bytes* (i.e. not DATA chunks or messages) which are still acceptable by the receiver. Note, that from the perspective of an implementer, this value is usually based on a heuristic: e.g. 1,000 bytes in messages of 1 byte require significantly more storage space than a single message of 1,000 bytes. See also [Ste07] for more details on this subject.

The need to possibly free buffer space occupied by already GapAck'ed DATA chunks leads to so-called *Reneging*. Figure 3.7 shows a SACK example, consisting of a CumAck for TSN #2 and GapAcks for TSN #4 to TSN #7. TSN #3 is still needed to perform an advancement of the CumAck. While CumAcks are obviously non-renegable, the receiver may revoke GapAcks, in this case: TSN #8. This may e.g. happen when the receive buffer gets too full to store earlier chunks being necessary for the next CumAck. Although reneging usually occurs rarely, since implementations try to avoid reneging for efficiency reasons, the sender must always be prepared to reneg GapAck'ed TSNs. That is, GapAck'ed chunks must remain stored in the send buffer, although these chunks are not currently being outstanding any more.

3.8 Congestion Control

SCTP uses a window-based congestion control with AIMD behaviour including appropriate byte counting (see Subsubsection 2.11.2.1) and fast recovery (see Subsubsection 2.11.2.2) for each path

independently. However, two differences apply:

1. SCTP has no specific MSS, since its chunks have a variable size. Instead of a MSS, the congestion control for path P uses the MTU of path P, denoted as MTU_P.

2. Since SCTP uses DATA chunks as its segments, which are created during segmentation and are therefore fixed in size (see Section 3.6), the minimum useful congestion window setting is one smallest path MTU. In contrast, TCP uses bytes as segments; its congestion window may be a fraction of one MSS.

That is, on α newly acknowledged bytes on path P in a fully-utilised congestion window c_P for slow-start threshold s_P, c_P is increased – as defined in [Ste07, Subsubsection 7.2.1 and Subsubsection 7.2.2] – as follows:

$$c_P = c_P + \begin{cases} \min\{\alpha, \mathrm{MTU}_P\} & (c_P \leq s_P) \\ \mathrm{MTU}_P & (c_P > s_P \wedge p_P \geq c_P) \end{cases}.$$

The variable p_P is the "partially acknowledged" counter for path P in congestion avoidance mode (see Subsubsection 2.11.2.1). In case of a retransmission (i.e. fast or timer-based) on path P, s_P and c_P are reduced – as defined in [Ste07, Subsubsection 7.2.3] – as follows:

$$\begin{aligned} s_P &= \max\left\{c_P - \frac{1}{2} * c_P, 4 * \mathrm{MTU}_P\right\}, \\ c_P &= \begin{cases} s_P & \text{(Fast Retransmission)} \\ \mathrm{MTU}_P & \text{(Timer-Based Retransmission)} \end{cases}. \end{aligned}$$

The RTO is dynamically calculated for each path separately, as described in Subsection 2.11.4. Furthermore, RFC 4960 introduces lower and upper bounds – denoted as *RTO.Min* and *RTO.Max* – in [Ste07, Subsubsection 6.3.1]. That is, whenever the computed RTO is less than RTO.Min, it is rounded up to RTO.Min. Analogously, whenever the computed RTO exceeds RTO.Max, it is rounded down to RTO.Max. The default settings are RTO.Min=1 s and RTO.Max=60 s (as defined in [Ste07, Section 15]). Further details on the SCTP congestion control can be found in [Ste07, Section 7].

SCTP also supports ECN (see Subsection 2.11.5); the exact behaviour is described in [Ste07, Appendix A]. Since ECN is not relevant within the context of this thesis, a further introduction is omitted here.

3.9 Burst Mitigation

In certain situations, a SACK may newly CumAck or GapAck a larger amount of TSNs, corresponding to multiple MTU-sized packets. This may e.g. happen when SACKs get lost, or because of timer-based retransmissions: the receiver may get some DATA chunks on an alternative, longer-delay path. According to the SACK handling rules of SCTP (see [Ste07, Subsection 6.7]), delayed acknowledgement defines that the path for sending a SACK is the last DATA chunk reception path. If some consecutive SACKs take a long-delay path, a newly sent SACK over a low-delay path may arrive at the DATA chunk sender first. This SACK, of course, newly acknowledges all DATA chunks, including the DATA chunks which have triggered the – yet not arrived – SACKs on the long-delay path. Due to the sudden fall in the number of *Outstanding Bytes* (i.e. the amount of yet unacknowledged data), the sender may immediately fill the gap between outstanding bytes and congestion window by

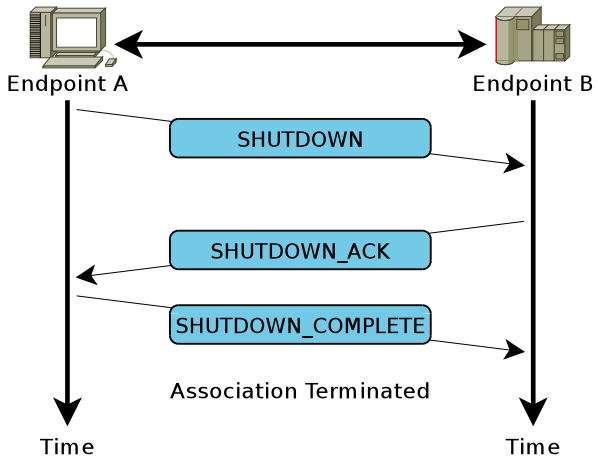


Figure 3.8: The Chunk Sequence of the SCTP Association Teardown

sending new DATA chunks. Of course, this leads to a burst of newly transmitted data – and possibly to congestion within the network.

[Ste07, Subsection 6.1] suggests to apply so-called *Burst Mitigation* (i.e. this feature is optional): before sending out new DATA chunks on path P, the sender should adapt its congestion window c_P:

$$c_P = \begin{cases} \text{Outstanding}_P + \text{MaxBurst} * \text{MTU}_P & (\text{Outstanding}_P + \text{MaxBurst} * \text{MTU}_P < c_P) \\ c_P & (\text{else}) \end{cases}.$$

MaxBurst denotes the maximum number of packets which may be sent in a burst. The suggestion from [Ste07, Section 15] is *MaxBurst*=4. That is, the sender must always utilise its congestion window c_P with outstanding bytes on path P (denoted by Outstanding_P), with a tolerance of four times the MTU; otherwise, it will be reduced. Due to this behaviour, the suggested mechanism is denoted as "Use It or Lose It" by [AB05].

3.10 Association Teardown

At the end of an SCTP communication, an association is normally terminated by a three packet exchange – as shown in Figure 3.8 – based on the *SHUTDOWN*, *SHUTDOWN_ACK* and *SHUTDOWN_COMPLETE Chunks*. This is the reliable form of an association teardown. It ensures that all sent user messages are received by the peer.

SCTP also provides the possibility of a "hard" break of an association, terminating the association immediately and taking message loss into account. This is performed by an *ABORT Chunk*.

3.11 Protocol Extensions

As explained in Section 3.2, SCTP can be extended easily by adding new chunk types. Multiple protocol extensions are existing – some as RFCs, some still as Internet Drafts – for additional features.

3.11.1 Chunk Authentication

The *Chunk Authentication* extension defined in [TSLR07] as RFC 4895 provides the possibility to authenticate selected types of chunks, by using a signature key combined of a key negotiated at association setup (i.e. being transferred as part of the handshake procedure; see also Section 3.3) and an optional pre-shared key (i.e. not transferred as part of the association). Chunk authentication can be applied to avoid attacks based on modification of the chunks of an association. It protects the integrity and authenticity of the chunks, but not their confidentiality.

3.11.2 Dynamic Address Reconfiguration

[SXT+07] defines the *Dynamic Address Reconfiguration* extension as RFC 5061. It allows to add and remove Network Layer addresses during association lifetime. Furthermore, it can notify a peer to set a certain path as primary path. A particular use case for dynamic address reconfiguration is endpoint mobility, i.e. it can be used to provide a seamless handover without the need of any support by the underlying Network Layers.

Since dynamic address reconfiguration would allow an attacker to hijack an association (by adding its own address and removing all other ones), the usage of the chunk authentication extension described in Subsection 3.11.1 is mandatory.

3.11.3 Partial Reliability

The *Partial Reliability* extension defined in [SRX+04] as RFC 3758 provides the possibility of so-called *Partially Reliable Transfer*, i.e. an intermediate step between unreliable and reliable transfer (see Section 2.9). According to a configurable *Retransmission Policy* for each message using partially reliable transfer, retransmissions are tried. If these retransmissions are not successful, the receiver side is finally told to ignore the missing message(s) and go ahead. This is realised by the *FORWARD_TSN Chunk*, containing a new cumulative TSN up to which all former messages are to be ignored.

A retransmission policy is completely realised at the sender side. While [SRX+04] only defines a limitation of the message lifetime, other variants – like limiting the number of retransmission trials – are also possible and used by some implementations (see also [DRS+11]).

3.11.4 Stream Reset

At association setup, the number of streams in each transport direction is negotiated. Furthermore, the SSN of each stream starts at zero. Some kinds of applications need to be able to increase the number of streams, as well as to reset the SSN of a stream, during association lifetime. These functionalities are provided by the *Stream Reset* extension defined in [STL12] as RFC 6525.

3.11.5 Non-Renegable Selective Acknowledgement

GapAcks in SACK chunks are renegable, i.e. GapAck'ed DATA chunks must remain in the send buffer until acknowledged by a CumAck. In order to non-renegably GapAck DATA chunks, the *Non-Renegable Selective Acknowledgement* (NR-SACK) extension defined in [EAN+11] as Internet Draft introduces the so-called *NR-SACK Chunk*. It is analogue to the SACK chunk described in Section 3.7, but further adds non-renegable GapAck blocks. DATA chunks being acknowledged by a

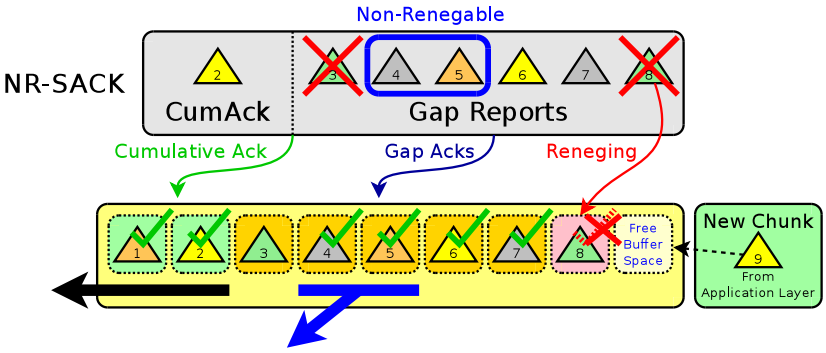


Figure 3.9: A Non-Renegable Selective Acknowledgement Example

non-renegable GapAck may be deleted from the send buffer immediately. NR-SACK therefore saves send buffer space.

Figure 3.9 provides an example: the CumAck for TSN #2 cumulatively acknowledges the TSN #1 and TSN #2. Furthermore, TSN #4 and TSN #5 are non-renegably GapAck'ed and their corresponding DATA chunks may also leave the send buffer – without waiting for TSN #3. However, the DATA chunks corresponding to TSN #6 and TSN #7 must still wait – they are normally GapAck'ed (i.e. renegable). Furthermore, TSN #8 has been reneged. An evaluation of NR-SACK and further details can be found in [NEY+08].

3.11.6 SACK Immediately

Delayed acknowledgement (see Subsubsection 2.9.3.1) improves the transport efficiency, but introduces additional delay. In some cases, it may be useful that a SACK for a DATA chunk is not delayed, but immediately sent. For example, if a DATA chunk is the last one of a sequence, there are no further DATA chunks to wait for. The *SACK Immediately* extension defined as Internet Draft in [TRS11] defines the "SACK Immediately" bit (I-bit) for the Flags field of the DATA chunk (see also Section 3.6). If I=1, the receiver is requested to send a SACK chunk without delay. An evaluation of this extension can be found in [Rün09].

3.11.7 Secure SCTP

In order to also protect the confidentiality of SCTP messages – in addition to integrity and authenticity as also provided by chunk authentication (see Subsection 3.11.1) – the *Secure SCTP* extension defined in [HDU12] as Internet Draft adds encryption and authentication mechanisms to SCTP. A detailed introduction and evaluation is provided in [HRUT07, Unu05, URJ04].

3.11.8 Packet Drop Reporting

Corrupted SCTP packets are simply dropped. However, since lost DATA chunks are assumed to be the result of congestion (see Subsection 2.11.2), the congestion control reduces the congestion window – which leads to a reduced throughput. The *Packet Drop Reporting* extension defined as Internet Draft in [SLT12] allows a receiver, as well as intermediate systems (e.g. satellite routers), to report a corrupted SCTP packet to its sender, in form of a *PKTDROP Chunk*. That is, it realises a

negative acknowledgement mechanism, as introduced in Subsubsection 2.9.3.5. Then, the sender may retransmit the corresponding packet as soon as possible, without reducing the throughput. A detailed evaluation of this extension is provided by [RTR09].

3.11.9 "Potentially Failed" Path State

A performance improvement on path failures can be achieved by the *Potentially Failed Path State* extension, which is proposed in [NEA+08] and defined as Internet Draft in [NN11]. When a path is considered unreliable, it enters the "Potentially Failed" state. In this state, it is not used for any new data transmissions, thus avoiding a performance degradation until the failed path gets actually deactivated by a timeout. Some further details on SCTP path failure handling and its performance are provided by [FJQ+08, JRT02].

3.11.10 Concurrent Multipath Transfer

The concurrent multipath transfer extension, which is shortly denoted as CMT-SCTP, provides the CMT functionality which will be introduced in detail in Section 4.2. CMT-SCTP is defined as Internet Draft in [ABD+12].

3.12 Compatibility and Interoperability

Besides the protocol extensions defined in Section 3.11, there are some further documents on SCTP, defining mechanisms and procedures for compatibility and interoperability with other protocols and systems.

3.12.1 Application Programming Interface

The *Application Programming Interface* (API) of SCTP is defined in [STP+11] as RFC 6458. This document also contains the API definitions for most of the protocol extensions described in Section 3.11. An introduction to SCTP programming can be found in [SX01].

3.12.2 UDP Encapsulation

The Internet Draft [TS12] defines the encapsulation of SCTP packets into UDP packets. This mechanism may be applied to traverse firewalls not allowing SCTP traffic, or to use SCTP on non-SCTP-capable operating systems. For example, UDP encapsulation is applied by [DZB+10, DR07] to run SCTP applications in the PLANETLAB – a platform only allowing UDP and TCP packets.

3.12.3 Checksum Offloading

In comparison to Internet-16 or Adler-32, the computation of CRC-32C checksums requires significantly more CPU power (see also Subsection 2.5.1). However, new NIC chipsets – like the Intel 82576 chipset – provide the capability to compute CRC-32C sums in hardware, leading to a relief of the CPU. Obviously, this feature of advanced NIC chipsets may be used by SCTP implementations to perform so-called *Checksum Offloading*. That is, SCTP checksums for outgoing and incoming SCTP packets are calculated by the NIC, instead of using the CPU.

An overview, and further details on the SCTP protocol extensions and the ongoing work on SCTP standardisation, have also been published in [DRS+11].

Feature	Linux	FreeBSD	MacOS	Windows	Solaris	sctplib	OMNeT++
Standard SCTP (RFC 4960)	yes	yes	yes	yes	yes	yes	yes
Explicit Congestion Notification	yes	yes	yes	yes	no	no	no
Chunk Authentication	yes	yes	yes	yes	yes	no	yes
Dynamic Address Reconfiguration	yes	yes	yes	yes	yes	exp.	yes
Partial Reliability	yes	yes	yes	yes	yes	yes	yes
NR-SACK	no	yes	yes	yes	no	no	yes
SACK Immediately	no	yes	yes	yes	no	no	yes
Secure SCTP	no	no	no	no	no	exp.	no
Stream Reset	no	yes	yes	yes	no	no	yes
Packet Drop Reporting	no	yes	yes	yes	no	no	yes
"Potentially Failed" Path State	no	yes	yes	yes	no	no	no
CMT-SCTP	no	yes	yes	yes	no	no	yes
UDP Encapsulation	no	yes	yes	yes	no	yes	no
SCTP API	yes	yes	yes	yes	mostly	yes	(not useful)
Checksum Offloading	yes	yes	no	no	yes	no	(not useful)

Table 3.1: An Overview of SCTP Implementations and Supported Features

3.13 Implementations

Table 3.1 provides an overview of the features provided by the major SCTP implementations, as of March 2012:

Linux Kernel SCTP in the Linux kernel 3.3[2],

FreeBSD Kernel SCTP in the FreeBSD release 9.0[3],

MacOS X Kernel SCTP for Apple MacOS 10.7[4] (based on the FreeBSD kernel implementation),

Windows Kernel SCTP for Microsoft Window XP, Vista and 7[5] (based on the FreeBSD kernel implementation),

Solaris Kernel SCTP in OpenSolaris 2009.06[6],

SCTPLIB Userland SCTP stable release 1.0.11[7], as well as

OMNeT++/INET Simulation Model development version[8] (to be described in detail in Chapter 5).

3.14 Application Scenarios

In the following, some important SCTP application scenarios will be introduced shortly, in order to show where the features of SCTP and its extensions are used.

[2] ftp://ftp.kernel.org/pub/linux/kernel/v3.0/.
[3] http://www.freebsd.org/.
[4] http://sctp.fh-muenster.de/sctp-nke.html, not officially supported by Apple.
[5] http://www.bluestop.org/SctpDrv/, not officially supported by Microsoft.
[6] http://www.opensolaris.org/.
[7] http://www.sctp.de/sctp.html; "experimental" features only available in experimental versions.
[8] See [DBPR10b]; public version to be released soon.

3.14.1 SS7 over IP Networks

The initial motivation for SCTP has been the transport of telephone signalling over IP networks, based on the *Signalling System No. 7* (SS7) introduced by [ITU93]. Such telephone signalling has very strict availability requirements, e.g. in order to still handle emergency calls even when some signalling components fail. [Jun05, Subsection 2.1.3] describes these requirements in detail. Particularly, the SS7 transport requires support for multi-homing, multi-streaming, datagram-oriented transfer (i.e. preservation of message framing) and per-message configurable ordered/unordered delivery. Clearly, TCP has not been able to fulfil the requirements. This has led to the standardisation of SCTP by the IETF SIGTRAN WG (see also Subsubsection 2.12.3.1).

3.14.2 IP Flow Information Export

Another SCTP-based application is the IP Flow Information Export (IPFIX) architecture defined in [SBCQ09] as RFC 5470, with the IPFIX protocol defined in [Cla08, QBC+08, TB08] as RFCs 5101 to 5103. IPFIX provides the export of information about flows running through a network, which are collected by so-called *Observation Points* (e.g. routers or special monitoring devices), to so-called *Collectors*. A collector may use this information for accounting, billing or debugging purposes.

IPFIX makes use of SCTP because of its reliability features. Furthermore, the partial reliability extension (see Subsection 3.11.3) may be applied to avoid overload situations: if the current amount of generated flow information cannot be exported to a collector any more, the partial reliability extension may drop old information in favour of new data. That is, the export is not blocked to reliably transfer old data (which may be out of date already). Instead, new (i.e. up to date) flow information may be sent. More details on IPFIX and SCTP can be found in [Cla08, Subsection 10.2].

3.14.3 Reliable Server Pooling

Another notable SCTP application is *Reliable Server Pooling* (RSerPool), which has been standardised by the IETF RSerPool WG (see Subsubsection 2.12.3.1) in [LOTD08, SXST08a, XSS+08, SXST08b, DT08, DM09] as RFCs 5351 to 5356 and RFC 5525. It provides a lightweight framework for server redundancy and session failover, in order to support availability-critical applications as well as load balancing.

Since RSerPool is used for the simulation tool-chain described in Appendix B, a brief overview will be presented in Appendix A. More detailed introductions to RSerPool are provided by [Dre11, DR09, DR08b, Dre07].

3.14.4 Further Application Scenarios

As described in [DRS+11], SCTP provides the same service as TCP, plus a set of advanced features to utilise the enhanced capabilities of modern IP networks and to support increased application requirements. That is, SCTP may be deployed in all application cases where TCP is currently used. By just replacing the Transport Layer, from TCP to SCTP, applications may already take benefit of features like multi-homing, dynamic address reconfiguration, the enhanced security of 4-way handshake and verification tag as well as the improved detection of data corruption by CRC-32C. With increasing maturity of SCTP implementations, more and more applications may use SCTP in the future, as suggested by [DR08c].

3.15 Summary

In this chapter, the core SCTP protocol as well as the existing protocol extensions have been introduced. Furthermore, a brief overview of available SCTP implementations and SCTP application scenarios has been given.

Chapter 4

Multipath Transfer

This chapter introduces multipath transfer, and in particular the corresponding extension for SCTP. Here, the state of the art *before* begin of this habilitation project is presented. Optimisations and extensions which are the result of this project are described together with the evaluations in Chapter 7 and Chapter 8. Furthermore, an overview of alternatives and related work to SCTP-based multipath transfer is presented.

4.1 Introduction

As motivated in Section 1.1, the existence of multiple addresses per endpoint may lead to multiple paths among endpoints. It is therefore desirable to apply so-called *Load Sharing* to distribute traffic among the – possibly disjoint – paths, in order to improve the application payload throughput. Approaches for load sharing may be applied on different layers of the OSI reference model (see Subsubsection 2.2.3.1). The following short overview of the approaches on the different layers uses the terminology which has been defined formally in Section 2.1.

4.1.1 Data Link Layer Approaches

The application of multipath transfer on the Data Link Layer is quite straightforward: multiple links are bundled to appear to the Network Layer as a single, high-capacity one. That is, data is split up appropriately (e.g. by packets, bytes or bits), transferred over the links and joined at the peer side. Clearly, approaches based on the Data Link Layer are specific for one transfer technology.

The approach of multipath transfer on the Data Link Layer is commonly used for ISDN (see Subsection 2.12.1 and in particular [Tan96, Section 2.5]), which allows to bundle two 64 Kbit/s channels to a 128 Kbit/s channel.

4.1.2 Network Layer Approaches

The idea of applying load sharing on the Network Layer, which is independent of the underlying transfer technologies, is already quite old. The approach of *Dispersity Routing* by [Max75] splits up packets into sub-packets, which are sent over different paths. Also, it distinguishes between redundant and non-redundant dispersity routing: the redundant variant sends copies of the sub-messages on other paths, in order to increase the probability that the whole packet can be reassembled, even when some sub-packets get damaged or lost. Due to evolution of the network technologies, the idea of dispersity routing has become less useful and more difficult to implement, as it is reasoned by [Max07] in detail.

Today, the common approach of load sharing on the Network Layer is to distribute whole packets to different paths. Since the window-based congestion control (as described in Subsection 2.11.2) is based on the assumption that a path usually uses the same trail, packet reordering is considered as a sign of congestion. Therefore, the relevant approaches typically use a hash over parts of the Network Layer PCI (e.g. destination and source IP addresses) to define a mapping of packet flows to trails. Practical solutions for such a mapping are introduced by [TH00, Section 4]. They are commonly applied for so-called *Equal-Cost Multi-Path* (ECMP) routing, which means that a routing protocol computes multiple trails. Then, routers distribute traffic among all trails having the same least cost. One possible approach for ECMP is defined in [Hop00] as RFC 2992.

Multipath routing is also an interesting approach in the context of sensor networks. Here, data is primarily sent over multiple paths for redundancy reasons, in order to improve the reliability of a data transfer. However, newer approaches – such as a system for weather measurements presented by [RMMG11] – also consider load sharing, in order to improve the data throughput. An overview of approaches for ad-hoc networks is presented by [DV06, TH01], with a focus on the efficient handling of frequent topology changes.

Of course, from the perspective of the Transport Layer, a certain Transport Layer flow – like a TCP connection or an SCTP association – uses the same path on the Network Layer. That is, load sharing on the Network Layer results in an *Inter-Flow Load Sharing*: only packets of *different* Transport Layer flows may take distinct paths. This also means that the QoS characteristics of different Transport Layer flows (i.e. throughput, delay, etc.; see also Section 2.4) may vary, e.g. two TCP connections – one via a slow, the other one via a fast path – will observe dissimilar bandwidths. The end-to-end effects of Internet path selection are examined in detail by [SCH+99]. Therefore, inter-flow load sharing is also often denoted as *Load Spreading*.

A Network Layer approach for multipath transfer may also hide the existence of multiple paths from the Transport Layer, in order to keep the corresponding protocols unmodified. This approach – denoted as identifier/locator split – will be introduced in some more detail in Section 4.4.

4.1.3 Transport Layer Approaches

The idea of inter-flow load sharing, as applied by the Network Layer approaches, may also be transferred to the Transport Layer. As example, the approach of Multipath Aware TCP by [DZT06] lets an adapted TCP protocol choose the outgoing path for each connection separately, based on up-to-date path information (RTTs and packet losses) collected by the TCP protocol itself.

Intra-flow load sharing is the more important and interesting approach on the Transport Layer. Unlike inter-flow load sharing, it allows to utilise the bandwidths of multiple paths to enhance the application payload throughput of a *single* Transport Layer flow. Notable approaches are the following:

- The Reliable Multiplexing Transport Protocol by [MK01] applies load sharing among the paths of a transport connection to improve the throughput of bulk data transfers over low-speed wireless access technologies. The protocol is rate-based, i.e. it behaves very differently from the window-based congestion control (see Subsection 2.11.2) applied in the Internet.

- Parallel TCP by [HS02] applies the striping of a connection to TCP-like so-called mircoflows over different paths. The striping is based on the bandwidth of each path P, which is computed as the ratio $\frac{c_P}{\mathrm{RTT}_P}$ of congestion window c_P and round-trip time RTT_P (see also Subsubsection 2.9.2.3). A control component handles the interaction among the microflows.

- The mTCP approach by [ZLK+04] is similar to Parallel TCP, but with emphasis on the robustness of the data transfer against path failures. It also addresses the fairness issue on shared bottlenecks (see Subsection 1.2.2) by performing ICMP-based `traceroute` measurements (see [KR08, Section 1.4] and [Mal93, Section 1]), in order to try a detection of such bottlenecks. However, this measurement approach is only applicable in carefully set up networks. The Internet – containing e.g. IPv6 over IPv4 tunnels, firewalls, routers not supporting these measurements, etc. – prevents a reliable detection of shared bottlenecks by such measurements.

The described research approaches are not applied in practise. However, they have influenced the two approaches relevant in the currently ongoing IETF activities on multipath transport: CMT-SCTP as well as MPTCP. Since these approaches are also relevant in the context of this thesis, they will be introduced in some more detail in Section 4.2 (CMT-SCTP) and Section 4.3 (MPTCP).

4.1.4 Higher-Layer Approaches

Similar to the striping of data transfers among multiple paths on the Transport Layer, this functionality may also be applied on higher layers of the OSI reference model. However, the difficulty in this case is that – for efficient operation – a tight interaction with the Transport Layer is required. That is, the striping of flows via different paths particularly requires mechanisms to avoid that the flow and congestion control on one path may block the whole communication. Some notable approaches are the following:

- [HAN02] uses striping over multiple TCP connections – which are transported over the *same* path – in order to improve the throughput of a bulk data transfer. This approach mitigates the impact of non-congestion losses, but directly implies unfairness to concurrent standard TCP connections.

- [SBG00] applies striping similar to the approach by [HAN02], with the focus on using TCP implementations not being state-of-the art. That is, since the maximum achievable throughput with a sliding-window-based reliable transfer is limited by the bandwidth-RTT product, as described in Subsubsection 2.9.2.3, and old TCP implementations limit the maximum send window to typically at most 64 KiB (as explained in [JBB92, Subsection 1.1]), a striping approach on top of such TCP implementations can improve the performance. Therefore, this approach is of practical usefulness in situations where the problem source – which is the limitation of the TCP implementation – cannot be solved, e.g. when a proprietary operating system does not support state-of-the-art TCP.

- A similar striping approach is also used by [AKO97], with the focus on improving the throughput performance over a satellite link.

- Multi-Connection TCP by [Sch10] introduces a striping protocol on top of TCP, with some flow control extensions to the TCP protocol itself. Each sub-connection uses another path. That is, this approach extends the striping ideas by the possibility to use different paths. Paths may or may not be disjoint, leading to the fairness issue on shared bottlenecks, as described in Subsection 1.2.2.

The necessary cross-layer interaction for higher-layer striping approaches blurs the borders among the layers and makes the management complex. Particularly, from an implementer's perspective, the proposed approaches also need a high interaction between the kernel (which provides the transport

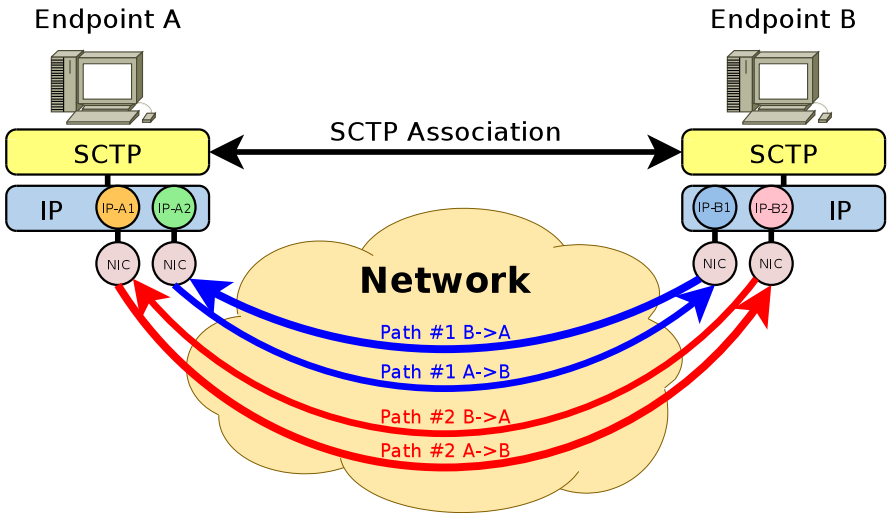


Figure 4.1: Applying Multipath Transfer for SCTP with the CMT-SCTP Extension

protocol) and the userland (which realises the striping functionality). Some more details on the problems of higher-layer approaches for multipath transfer are presented in [HS02, Section 4.3]. Since none of the described approaches has a practical relevance, a further discussion is omitted here.

4.2 CMT-SCTP – Multipath Transfer for SCTP

The application of multipath transfer on the Transport Layer seems to be the most practicable solution when intra-flow load sharing is desired. Applying SCTP as Transport Layer protocol, the usage of multipath transfer seems – at least at first sight – quite straightforward.

4.2.1 Basic Approach

Since SCTP supports multi-homing, an SCTP entity just has to distribute its DATA chunks among *all* of its paths, instead of using a designated primary path. The principle is illustrated in Figure 4.1. Clearly, this is also the approach of the *Concurrent Multipath Transfer* extension for SCTP suggested by [IAS06], which is further denoted as CMT-SCTP.

CMT-SCTP assumes that all paths are disjoint, i.e. the trails actually used for routing must be disjoint (see also Subsection 2.1.4). This requires a careful configuration of the routing within the network. Having disjoint paths, the SCTP congestion control – as introduced in Section 3.8 – may be reused without any change.

However, just modifying standard SCTP to send its DATA chunks over multiple paths leads to a very poor throughput performance. In order to perform multipath transfer efficiently, three additional optimisations are necessary and included in CMT-SCTP as described by [IAS06]. These mechanisms will be introduced in the following, based on examples which have also been published in [DBPR10b].

4.2.2 Split Fast Retransmission

When sending over multiple paths, DATA chunks may overtake each other due to different path delays. An example is illustrated in Figure 4.2: sending TSN #9 over Path #1 and TSN #10 to TSN #12 over

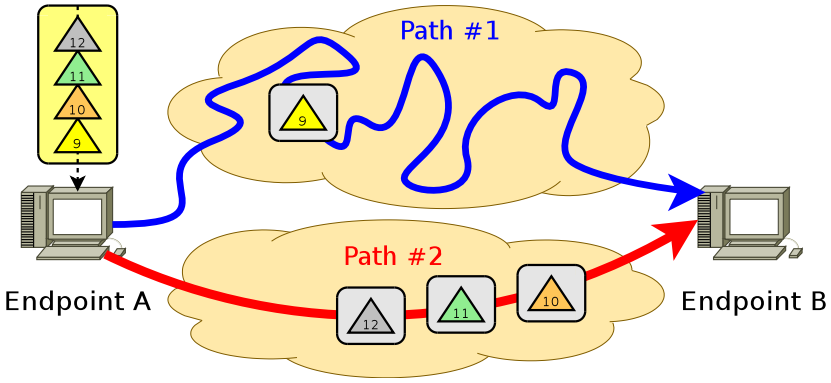


Figure 4.2: The Challenge of Fast Retransmission with CMT-SCTP

Path #2, the receiver may first see the TSN #10 to TSN #12, while TSN #9 is still on its travel through the network. Consequently, it will notice three out-of-sequence TSNs and acknowledge each one – as described in Section 3.7 – with a CumAck for TSN #8 (i.e. the last in-sequence TSN) and corresponding GapAcks for the already-received TSNs #10 to #12. Then, the sender sees three CumAcks for the same TSN and therefore triggers a fast retransmission – which requires a retransmission of TSN #9 and leads to a reduction of the congestion window. When finally the original TSN #9 arrives at the receiver, the following retransmitted chunk – being a duplicate – can be ignored (i.e. network bandwidth has been wasted), and the congestion window has to grow again (i.e. time will be wasted).

The solution to this problem – denoted as *Split Fast Retransmission* – is reasonably simple: the SACK handling has to take care of the individual paths. For each transmitted DATA chunk, the path on which it has been sent is remembered. On reception of a SACK, the path P of a chunk with TSN τ_{Missing} reported as missing is checked. TSN τ_{Missing} is only assumed as missing if τ_{Missing} is smaller than the TSN of the highest successfully acknowledged chunk $\tau^{P}_{\mathrm{HighestAckedOnPath}}$ on path P. Since both, the chunks with TSNs τ_{Missing} and $\tau^{P}_{\mathrm{HighestAckedOnPath}}$ have been transmitted on path P, and the later TSN $\tau^{P}_{\mathrm{HighestAckedOnPath}}$ has been received successfully, TSN τ_{Missing} is probably lost. Otherwise, there is nothing to do.

In the example above, this means that TSN #9 would be fast retransmitted if a new TSN #13 on Path #1 would have been acknowledged while still awaiting TSN #9.

4.2.3 Congestion Window Update for CMT

When a new CumAck has been received, the congestion window may grow, as explained in Section 3.8. On the reception of SACKs with the same CumAck'ed TSN, the congestion window is left unchanged. An example is illustrated in Figure 4.3, where Endpoint A sends the chunks with TSN #13 and TSN #14 over Path #1 to Endpoint B. The chunks with TSNs #15 to #18 use Path #2. Due to congestion and reordering, the chunks with TSNs #14 and #17 do not arrive before a SACK chunk is sent back to Endpoint A. This SACK contains a CumAck for TSN #13 (i.e. all chunks up to TSN #13 have been received successfully) and GapAcks for the TSNs #15, #16 and #18.

If Endpoint A applies the congestion window update strategy of standard SCTP, it sees an increase of the CumAck to TSN #13. That is, the SACK has acknowledged one TSN on Path #1 and the sender may grow the congestion window of Path #1. However, the congestion window on Path #2 would not increase, since TSN #14 – which has been sent on the *other* path – is still unacknowledged. Also,

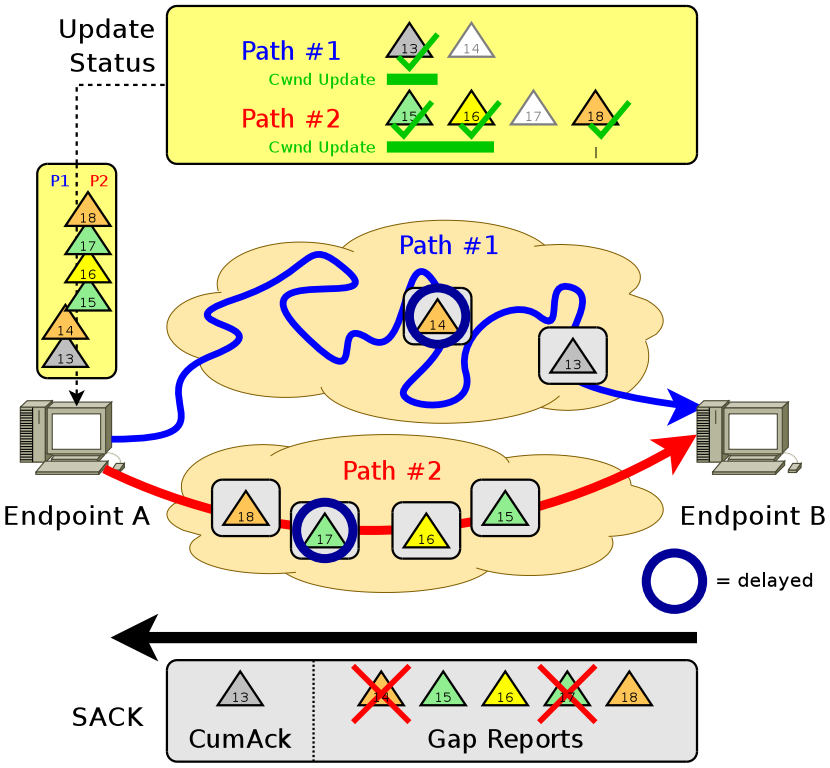


Figure 4.3: The Challenge of Congestion Window Updates with CMT-SCTP

the congestion window handling of later SACKs will only take care of *newly* acknowledged TSNs. That is, when the missing TSNs are eventually acknowledged later, the window must not grow. This restriction is necessary to avoid sudden increases of the congestion window leading to bursts of newly sent data (see also Subsubsection 2.11.2.1).

In order to improve the efficiency of CMT-SCTP, the *Congestion Window Update for CMT* strategy takes the existence of different paths into account. On reception of a SACK, it looks for the earliest outstanding TSN on each path. For the example in Figure 4.3, this is TSN #14 on Path #1 and TSN #17 on Path #2. Therefore:

1. On Path #1, the SACK has newly acknowledged the TSN #13 – i.e. one new TSN has been acknowledged and the congestion window on Path #1 may grow accordingly.

2. On Path #2, there are no outstanding TSNs smaller than TSN #15; the TSNs #15 and #16 have been acknowledged newly. Therefore, the congestion window on Path #2 may also grow accordingly. Note, that TSN #17 cannot be assumed as lost here yet – it may just be reordered. On reception of the – by default – third SACK reporting it as missing, a fast retransmission would be scheduled.

Since the CMT congestion window update strategy maintains a virtual CumAck for each path, it is also denoted as *PseudoCumAck*. A further improvement – denoted as *Congestion Window Update for CMT, version 2* – also takes care of TSNs being retransmitted. When there is a retransmission of a TSN on a path differing from the original one, the update strategy cannot reliably keep track of the PseudoCumAck of either path. Therefore, the improved version makes a distinction between TSNs

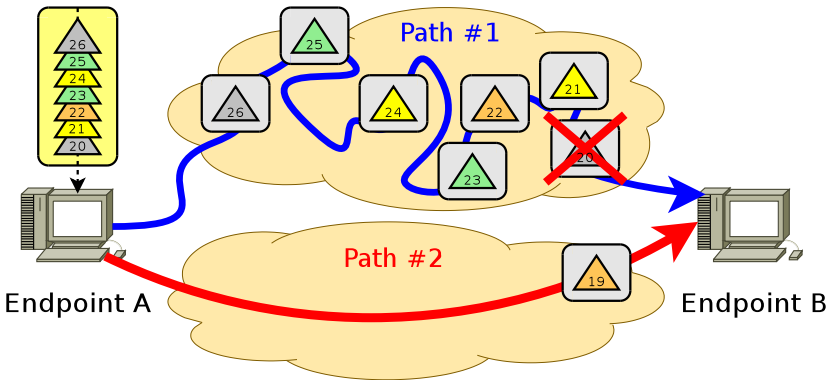


Figure 4.4: The Challenge of Delayed Acknowledgement with CMT-SCTP

which have been transmitted only once, and TSNs having been retransmitted. A second PseudoCumAck – denoted as *Retransmission PseudoCumAck* – is maintained; whenever one of these two PseudoCumAcks is increased, the congestion window may grow according to the procedures explained in Subsubsection 2.11.2.1.

4.2.4 Delayed Acknowledgement for CMT

When DATA chunks arrive in sequence, SCTP does not immediately send a SACK chunk for acknowledgement. Instead, it applies delayed acknowledgement – as described in Section 3.7 – in order to reduce overhead traffic. But for reordered chunks, a SACK chunk is sent immediately. By default, the reception of three SACKs for the same TSN triggers a fast retransmission to retransmit a lost DATA chunk; a loss on reordering is likely in the non-CMT case. However, in case of CMT, reordering is frequent and usually does *not* imply a loss. But each received out-of-sequence DATA chunk would require the immediate transmission of a SACK chunk. Unnecessary fast retransmissions are avoided by split fast retransmission already (see Subsection 4.2.2), but the increased SACK traffic overhead remains.

In order to overcome this inefficiency for CMT, the *Delayed Acknowledgement for CMT* strategy simply delays *all* SACK transmissions. However, this would also delay the recovery of a real packet loss, which is triggered by the SACKs. An example is provided in Figure 4.4:

1. TSN #20 is lost. No SACK chunk will be transmitted.
2. The TSNs #21 and #22 are received and acknowledged by a SACK chunk. Endpoint A recognises the missing TSN #20 for the first time.
3. The TSNs #23 and #24 are received and another SACK chunk is sent. Endpoint A recognises the gap for the second time.
4. The TSNs #24 and #25 are received and SACK'ed. Endpoint A sees the gap for the third time. This is the default threshold for a fast retransmission of the lost DATA chunk.

That is, the loss is detected after six more chunks have been sent. Using standard SCTP behaviour, it would have been detected after only three chunks (the third SACK would have been sent after receiving the out-of-sequence TSN #23).

Delayed acknowledgement for CMT solves the problem by two steps: first, the receiver has to put the number of TSNs received since sending the previous SACK chunk into each SACK chunk. Then, the SACK handling procedure of split fast retransmission (see Subsection 4.2.2) for a missing TSN τ_{Missing} has to be modified. An incoming SACK is handled as follows:

- If all newly acknowledged TSNs have been transmitted over the same path P:
  - If there are newly acknowledged TSNs τ_L and τ_H so that $\tau_L < \tau_{\text{Missing}} < \tau_H$, the missing count is incremented by one; there may have been some reordering on the path P.
  - Else, if *all* newly acknowledged TSNs τ_N satisfy the condition $\tau_{\text{Missing}} < \tau_N$, the missing count is incremented by the number of TSNs reported in the SACK chunk; the corresponding DATA chunk is probably lost.
- Otherwise (i.e. there are newly acknowledged TSNs on different paths), the missing count is incremented by one (just like for standard SCTP).

For the example shown in Figure 4.4, applying delayed acknowledgement for CMT means that TSN #21 is GapAck'ed with a CumAck for TSN #19. Since this SACK chunk contains newly acknowledged TSNs on both paths, the missing counter for TSN #20 will be increased to 1. After receiving the second SACK chunk – containing the newly GapAck'ed TSNs #22 and #23 and a count of 2 TSNs since the last SACK – the missing counter for TSN #20 grows by 2 to 3. By default, this is the threshold for a fast retransmission. That is, this retransmission is triggered after three TSNs – which is as quickly as in a non-CMT scenario.

4.3 Multi-Path TCP – Multipath Transfer for TCP

Multi-Path TCP (MPTCP), which is introduced by [FRH+11], denotes an experimental multipath transfer extension for the TCP protocol. TCP – as described in Sebsection 2.13.3 – is a well-known and widely deployed protocol. During the now 30 years after its definition in [Pos81c] as RFC 793, it has been improved by various protocol extensions.

In contrast to SCTP, which is relatively new, the backwards compatibility to existing TCP implementations and middlebox devices is a very important design criteria of MPTCP. The term *Middlebox* in this context particularly denotes firewalls performing stateful packet inspection as well as routers providing Network and Port Address Translation, i.e. changes of Network and Transport Layer addresses (i.e. IP addresses and TCP ports) as defined in [EF94] as RFC 1631 as well as in [SE01] as RFC 3022, respectively. The need for interoperability implies that – from the perspective of a non-MPTCP-aware middlebox device – the packet flow of a single MPTCP path must look like a regular TCP connection. This leads to two fundamental differences in comparison to CMT-SCTP:

1. An *MPTCP Path* is defined by a tuple consisting of source and destination address (as being further explained in [FRHB11, Section 3]), in contrast to just the destination address as for SCTP (as defined in Subsection 3.4.1). This is necessary in order to let each path appear like a single standard TCP connection to middlebox devices.

2. Furthermore, each MPTCP path must use its own sequence number space, in order to mimic a contiguous segment sequence for the middlebox devices on the path.

Clearly, the first constraint implies that the number of MPTCP paths is much higher: $m * n$ paths for m local and n remote addresses, in contrast to only $m + n$ paths for SCTP. Also, the management of

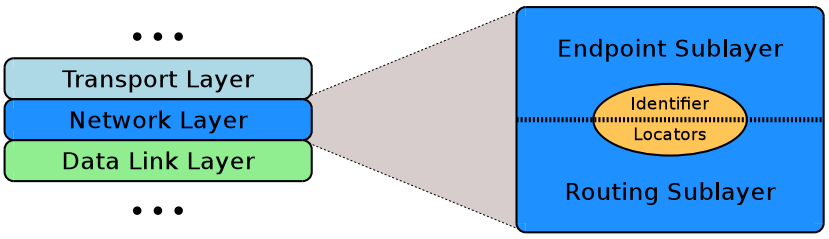


Figure 4.5: The Principle of Identifier/Locator Split on the Network Layer

multiple sequence number spaces introduces additional complexity. More details on MPTCP can be found in [BPB11, RBP+11].

4.4 Identifier/Locator Split – Multipath Transfer on the Network Layer

While Transport Layer solutions for load sharing require significant changes of the corresponding Transport Layer protocols, approaches based on the Network Layer may leave the higher layers untouched. The common approach for the Network Layer is denoted as *Identifier/Locator Split*; its principle is illustrated in Figure 4.5. The Network Layer is split up into two Sublayers:

Endpoint Sublayer This sublayer is responsible for the unique identification of an endpoint. The endpoint identification functionality is realised by *Identifiers*.

Routing Sublayer The responsibility of this sublayer is the forwarding of data; it resembles the functionalities provided by the "classic" Network Layer as described in Subsubsection 2.2.3.1. The addresses used for the routing are denoted as *Locators*.

While the upper layers only have to take note of the identifier, the extended Network Layer is responsible for translating the identifier into locator(s), in order to forward data over a useful Network Layer path. Since the upper layers are unaware of this translation, load sharing for the same Transport Layer flow is not possible – functionalities like window-based congestion control (see Subsection 2.11.2) would not work properly. That is, identifier/locator split approaches cannot perform intra-flow load sharing, but they can do inter-flow load sharing. Furthermore, the multi-homing functionality introduced by these approaches also implies an easy support for mobility (i.e. locator changes) without the need to adapt higher-layer protocols.

Two IETF standards for identifier/locator split are relevant:

- The *Host Identity Protocol* (HIP) is suggested by [JNM+04] and defined in [MN06] (architecture; as RFC 4423) and [MNJH08] (protocol; as RFC 5201). It introduces an identifier space – being based on a public key infrastructure – which defines *Host Identities*. HIP is defined for IPv4 as well as IPv6, i.e. the IP addresses are the locators. Clearly, upper-layer protocols must be capable to handle the host identities, which requires changes of the protocols.

- The *Shim6* protocol, which is suggested in [Sav06, dLB06] and finally defined in [NB09] as RFC 5533, provides an identifier/locator split approach for IPv6 only. The identifiers introduced by Shim6 – called *Upper-Layer Identifiers* – have a size of 128 bits, i.e. they can just be handled like IPv6 addresses (see also Subsection 2.13.2). Therefore, no changes of the upper-layer

protocols are necessary. This reduces complexity and makes a deployment of Shim6 much easier compared to HIP, but – of course – also restricts it to IPv6 only.

Since identifier/locator split does not provide intra-flow load sharing, these approaches are not discussed in more detail here. Further details on these approaches can e.g. be found in [SC05].

4.5 Summary

In this chapter, the idea of multipath transfer as well as possibilities to realise it on different layers of the OSI reference model have been presented. Furthermore, some important realisations have been introduced: CMT-SCTP and MPTCP on the Transport Layer as well as identifier/locator split – with HIP and Shim6 – on the Network Layer.

Chapter 5

The Simulation Environment

This chapter describes the simulation environment which has been used and extended, in order to perform a performance evaluation of CMT-SCTP. That is, it introduces the underlying discrete event simulation environment, its network protocols package as well as the CMT-SCTP model itself and the application model.

5.1 Introduction

In order to create a simulation environment for CMT-SCTP, it had clearly been useful to extend the SCTP model of an existing discrete event simulation environment. Three models in three different environments had been available:

1. The open source package NS-2[1] (Network Simulator 2) – as described by [FV10] – is a well-known and widespread simulation environment. Also, NS-2 includes an SCTP simulation model, which is shortly described by [HG02]. However, this model is unmaintained and therefore does not include state-of-the-art features of SCTP and its extensions.

2. Another alternative is the commercial, closed source OPNET MODELER[2], which is introduced in [SP03]. An SCTP model for the OPNET MODELER is described by [Jun05], it had been developed as part of a Ph.D. thesis at the University of Duisburg-Essen. However, due to high licensing costs of the underlying simulation environment, development had been ceased after the end of this project and the expiration of the OPNET MODELER licence. Therefore, the SCTP model had been out-of-date.

3. The third environment is the open source OMNET++[3] (Objective Modular Network Testbed in C++), as introduced in detail by [Var10]. Also, there had been a state-of-the-art and actively maintained SCTP model by [RTR08], which had in fact been a highly updated port of the OPNET MODELER model described above, developed as part of a Ph.D. thesis by [Rün09] at the University of Duisburg-Essen.

Clearly, since the OMNET++-based SCTP model had been state-of-the-art, and there had already been a lot of experience with the OMNET++ environment as result of former research work on

[1] NS-2: http://www.isi.edu/nsnam/ns/.
[2] OPNET MODELER: http://www.opnet.com/.
[3] OMNET++: http://www.omnetpp.org/.

RSerPool (see [Dre07]), the choice of this environment had been clear and straightforward. Therefore, the alternatives are not discussed in more detail here.

5.2 OMNeT++

In the following, a brief overview of the important basics of OMNeT++ is provided, in order to understand the following description of the IP network and SCTP models. Details on OMNeT++, including very detailed examples, can be found in [Var10]. An introduction to discrete event simulation in general is provided by [Got10].

Each OMNeT++ simulation model consists of a so-called *Network*. It is the root in a hierarchy of so-called *Modules*. Two types of modules are existing:

1. A *Simple Module* is atomic; its functionalities require implementation – in C++ language – by the developer of the module.

2. On the other hand, a *Compound Module* consists of at least one sub-module; each sub-module may again be a compound module or a simple module.

The interfaces between modules are denoted as *Gates*. A *Connection* links a gate to the gate of another module. The gate connections are used for the transport of *Messages* among the modules. Connections may use so-called *Channels* to introduce bandwidth limitations, delays, losses and errors[4] to messages. The global *Future Event Set* realises timers by allowing to schedule a message to a module itself, at a certain time stamp. As soon as this time stamp is reached by the simulation time, the message gets delivered. The definition of messages is performed in an OMNeT++-specific message definition language. Networks as well as modules with their gates, connections and channels are specified in the OMNeT++ NEtwork Description Language (NED).

A simulation – consisting of a definition of its network by an NED file and its compiled executable – can be parametrised by configuration settings read from a so-called .ini file (named due to its suffix). For example, an own .ini file may be created for each run, specifying the particular simulation run configuration including its random number generator seed. During a simulation run, two output files are generated:

1. The *Vector File* contains *Vectors*; a vector denotes a time series, e.g. the congestion window as tuples of recording time stamp and value. That is, vectors are frequently used to display time-based state changes.

2. The other file is the *Scalar File*, which contain *Scalars*. A scalar denotes a single result value, e.g. the average received throughput over the whole measurement time.

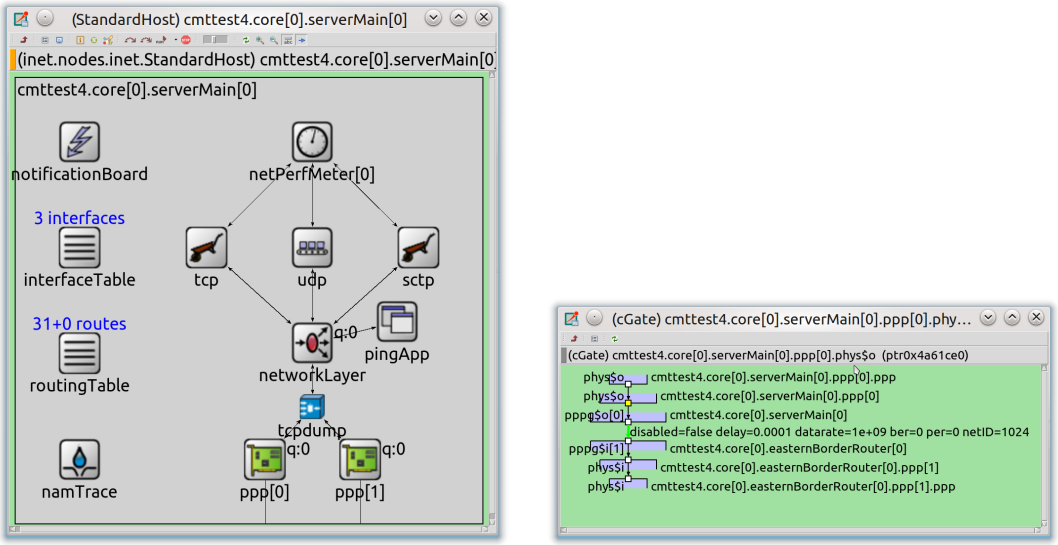
The OMNeT++ package (in the used release version 4.1) itself contains the full implementation of the discrete event simulation core, including message, NED and .ini file handling, as well as some example modules. However, it does not include IP network modules.

5.3 The INET Framework

The IP network modules of OMNeT++ are provided by a separate package, which is denoted as the INET Framework[5] and documented in [Var12]. Like OMNeT++, the INET Framework is

[4] A bit error is introduced by setting an error flag in the message object. The actual message remains unmodified; the receiving module has to take care of the error flag and implement appropriate reactions.

[5] INET Framework: http://inet.omnetpp.org/.



(a) StandardHost (b) Gates and Connections

Figure 5.1: An Instance of the StandardHost Compound Module and one of its PPP Interfaces

also open source. It provides models for the common protocols, particularly Ethernet (see Subsection 2.12.2), PPP (see Subsection 2.13.1), IPv4 and ICMP as well as IPv6 and ICMPv6 (see Subsection 2.13.2), UDP and TCP (see Subsection 2.13.3), as well as some test applications. The model for the SCTP protocol (see Chapter 3) had originally been developed separately, as described in [RTR08], but later been contributed to the standard INET FRAMEWORK package.

A particularly useful feature of the INET FRAMEWORK is its external interface, which is described in detail by [TRR08]. It makes it possible to exchange messages between the simulation and the real world. That is, using a real-time scheduler for the simulation core, external components (e.g. SCTP test programs on a FreeBSD machine) may communicate with simulated network components. This may e.g. be used to apply a complex virtual topology with specific QoS properties for tests and measurements between real components. Furthermore, it provides a powerful possibility for model validation: to an external component (e.g. using FreeBSD kernel SCTP), the SCTP implementation within the simulation just behaves like any other real SCTP implementation. Therefore, the simulation model has been successfully tested against the other relevant SCTP implementations (see Section 3.13 and also [DRS+11]) at the 9th SCTP Bakeoff[6], i.e. the official SCTP implementers meeting for interoperability testing. Unlike other SCTP simulation models (see Section 5.1), the SCTP model in the INET FRAMEWORK is therefore also a "real" SCTP implementation, which has furthermore been validated intensively.

Since the external interface provides code to translate between OMNET++ messages and real IPv4/IPv6 packets, a straightforward feature is provided by the *TCPDump* module: it allows components to dump packet traces in the well-known *Packet Capture* (PCAP) format, which is also used by TCPDUMP[7] (see also [Tan96]). These traces may be used for protocol analysis and debugging (to be explained in Section 6.4).

[6] 9th SCTP Bakeoff: http://www.interop.sctp.jp/.
[7] TCPDUMP: http://www.tcpdump.org/.

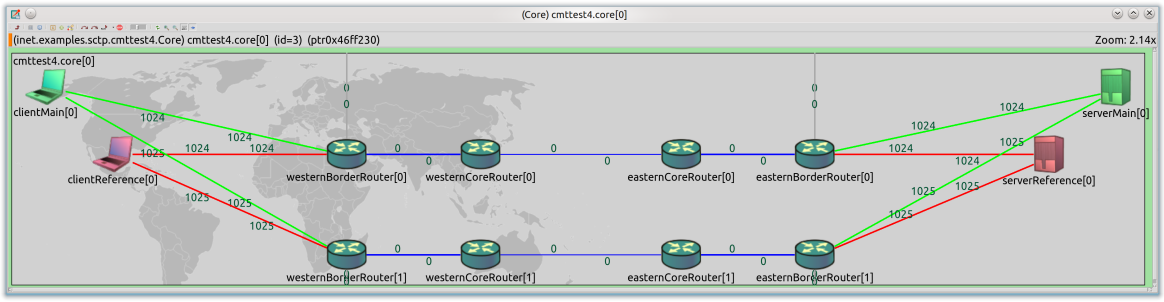


Figure 5.2: A Topology Consisting of StandardHost Instances

An important module of the INET FRAMEWORK is the *StandardHost* compound module, as depicted in Figure 5.1. Subfigure 5.1(a) shows an example instance of this module. It includes a Network Layer instance (*networkLayer*; here: IPv4) as well as instances of TCP, UDP and SCTP (*tcp*, *udp* and *sctp*, respectively). Like a real network host, it also contains a routing table (*routingTable*) and an interface table (*interfaceTable*). In the shown example, the node has two PPP interfaces: *ppp[0]* and *ppp[1]*. The configurations for both, interfaces and routing, may be read from a configuration file. *tcpdump* denotes the instance of the *TCPDump* module introduced above. It is located between Network Layer instance and interfaces and may be used to write an IP packet trace to a PCAP file.

The egress gate configuration of interface *ppp[0]* is shown in Subfigure 5.1(b). Its PPP instance is connected to a router, using a channel with a throughput (denoted by *datarate*) of 1 Gibt/s (i.e. $1 * 10^9$ bit/s and a delay (denoted by *delay*) of 0.0001 s (i.e. 0.1 ms), without bit errors (denoted by *ber* – bit error rate) or packet losses (denoted by *per* – packet error rate). The ingress side (not shown here) looks similar.

Using the *StandardHost* module, it becomes easy to create complex network topologies, as shown in the example in Figure 5.2. Each client, server and router is simply an instance of *StandardHost*; the number of interfaces, types and numbers of applications, etc. as well as the behaviour is simply a parameter setting of the corresponding instance. For example, a router is simply a *StandardHost* with IP packet forwarding enabled. The *StandardHost* instance shown in Subfigure 5.1(a) displays the server *serverMain[0]*. Instances of models are hierarchically labelled, with the network as root. In the example shown here, the network is named *cmttest4*. The topology shown in Figure 5.2 is the first instance of a compound module within an array named *core*, i.e. its name is *core[0]*. Then, the server *serverMain[0]*, which is the first instance in the array *serverMain* and whose *StandardHost* instance is presented in Subfigure 5.1(a), has the full name *cmttest4.core[0].serverMain[0]*. This naming is also used within the .ini file containing the parameters of a simulation run, i.e. each instance may be configured separately.

A more detailed introduction is omitted here, since this would greatly exceed the scope of this brief overview. An extensive documentation of the INET FRAMEWORK and its large library of modules is provided in [Var12].

5.4 The CMT-SCTP Model

To actually examine CMT-SCTP, the SCTP model introduced by [RTR08] – which is part of the INET FRAMEWORK – has been extended. This enhanced model – which is denoted as *CMT-SCTP Model*

Parameter	Functionality	Default Setting
cmtCCVariant	CMT and Congestion Control Variant	off
cmtUseSFR	Enable/Disable Split Fast RTX	true
cmtUseDAC	Enable/Disable Delayed Ack. for CMT	true
cmtCUCVariant	Congestion Window Update Strategy	pseudoCumAckV2
checkSackSeqNumber	Verify (NR-)SACK TSN Sequence	false
checkQueues	Verify Integrity of Chunk Bookkeeping	false
maxBurstVariant	Burst Mitigation Variant	useItOrLoseIt
maxBurst	Per-Path Maximum Burst Size (packets)	4
nrSack	Enable/Disable NR-SACK Extension	false
disableReneging	Enable/Disable Reneging	false
cmtSmartFastRTX	Enable/Disable Smart Fast RTX	true
cmtSackPath	SACK/NR-SACK Path Selection	smallestSRTT
cmtBufferSplitVariant	Buffer Splitting Strategy	none
cmtBufferSplittingUsesOSB	Buffer Splitting on Outstanding Bytes	false
cmtChunkReschedulingVariant	Check Rescheduling Strategy	none
cmtChunkReschedulingThreshold	Check Rescheduling Threshold	0.5
rpPathBlocking	Enable/Disable RP Path Blocking	true

Table 5.1: The Important CMT-SCTP Parameters of the SCTP Module

in the following – is also shortly introduced in [DBPR10b].

5.4.1 Added Features and Parameters

The existing SCTP module has been extended by CMT-SCTP, i.e. all SCTP-based INET simulation models could make use of CMT. In order to configure the CMT-SCTP functionality, the parameters summarised in Table 5.1 have been added. By default, CMT is turned off (i.e. cmtCCVariant set to "off"). Unless explicitly changing cmtCCVariant, the new SCTP module behaves exactly like the old, non-CMT model. The setting of cmtCCVariant="cmt" activates CMT-SCTP, as described in Chapter 4. The parameters cmtUseSFR and cmtUseDAC enable or disable the split fast retransmission (see Subsection 4.2.2) and the delayed acknowledgement for CMT (see Subsection 4.2.4) optimisations. The useful default is to turn on both, of course. cmtCUCVariant selects the congestion window update strategy (see Subsection 4.2.3): "normal" as for standard SCTP, "pseudoCumAck" (with PseudoCumAck) or "pseudoCumAckV2" (with PseudoCumAck and Retransmission PseudoCumAck – the useful default). The previously described four parameters realise CMT-SCTP as initially defined by [IAS06], i.e. CMT-SCTP before this habilitation project.

As part of this project, further features have been added. Their important *parameters* are shortly introduced here to complete the description of the model. However, the actual description of their *functionalities* follows in Chapter 7 and Chapter 8:

- checkSackSeqNumber and checkQueues enable ("true") or disable ("false") features to verify the TSN sequence of SACK chunks as well as of the DATA chunk bookkeeping structures (including statistics counters). These features are for debugging purposes only; they do not add actual protocol features but are highly useful for a researcher extending the model.

- The burst mitigation variants described in [AB05] for TCP have been adapted to SCTP (to be described in Subsubsection 7.9.4.3) and added to the model. The parameter maxBurstVariant selects the variant ("useItOrLoseIt" for Use It or Lose It – the suggested variant in [Ste07, Section 6.1], "congestionWindowLimiting" for Congestion Window Limiting, "maxBurst" for Max Burst). The maximum number of packets which may be sent in a burst may be limited by the parameter maxBurst (default is 4; recommendation from [Ste07, Section 15]).
- The NR-SACK extension (as described in Subsection 3.11.5) has been added. The option nrSack enables ("true") or disables ("false") it. Furthermore, the parameter disableReneging may be used to disable reneging completely ("true") or behaving like standard SCTP ("false").
- The parameter cmtSmartFastRTX activates ("true") or deactivates ("false") the smart fast retransmission functionality to be introduced in Subsection 7.7.3. Furthermore, cmtSackPath controls the choice of paths to send SACK/NR-SACK chunks on when using CMT. The setting "standard" uses the path of the last DATA chunk (like standard SCTP defined in [Ste07]); the default "smallestSRTT" applies smart SACK path selection (to be introduced in Subsubsection 7.9.4.2).
- cmtBufferSplitVariant configures the variant of buffer splitting ("none", "senderSide", "receiverSide", "bothSides"), which will be introduced in Section 7.6. Furthermore, the option cmtBufferSplittingUsesOSB defines whether buffered bytes ("false") or outstanding bytes ("true") are used for applying buffer splitting.
- cmtChunkReschedulingVariant sets the variant of chunk rescheduling ("none", "senderSide", "receiverSide", "bothSides"), to be introduced in Section 7.8. The corresponding blocking fraction threshold is set by the parameter cmtChunkReschedulingThreshold.
- Furthermore, additional settings for cmtCCVariant – defining different variants of CMT congestion control – are defined:
  - "off" deactivates CMT (default setting). All other settings turn CMT on.
  - "cmt" sets plain CMT-SCTP congestion control (as described in Subsection 4.2.1).
  - "cmtrp" sets CMT/RPv1 congestion control (to be defined in Subsubsection 8.3.1.1).
  - "cmtrpv2" sets CMT/RPv2 congestion control (to be defined in Sububsection 8.3.1.2).
  - "mptcp-like" sets MPTCP-like congestion control (to be defined in Subsection 8.3.2).
- Finally, rpPathBlocking controls the usage of RP path blocking (to be introduced in Section 8.4) for congestion controls based on resource pooling. "true" activates this feature, "false" turns it off.

5.4.2 Interaction with TCPDump Module and External Interface

Like the original SCTP model, the CMT-SCTP model also supports the external interface provided by the INET FRAMEWORK. The procedures to translate between OMNET++ message objects and real SCTP packets have been adapted appropriately, including support for the NR-SACK extension. This also means that the *TCPDump* module to write PCAP traces is supported. The latter is particularly useful to analyse an OMNET++-generated packet trace by using the packet trace analysis tool WIRESHARK, which is also used in the real network testbed. It will be introduced in more detail – together with the testbed environment – in Section 6.4. WIRESHARK has been intensively used for validating and debugging the model.

Parameter	Functionality	Default Setting
activeMode	Active Mode (true) or Passive Mode (false)	true
protocol	Transport Protocol (“SCTP”, “TCP” or “UDP”)	“SCTP”
localAddress	Local IP Address (“” for “*any*” Address)	“”
localPort	Local Port	9000
remoteAddress	Remote IP Address (Active Mode only)	“”
remotePort	Remote Port (Active Mode only)	9000
primaryPath	Primary Path (SCTP only)	“”
outboundStreams	Number of Outbound Streams (SCTP only)	1
maxInboundStreams	Maximum Number of Inbound Streams (SCTP only)	16
connectTime	Absolute Time of Connection Establishment	0s
startTime	Relative Time of Payload Data Transfer Start	1s
resetTime	Relative Time of Measurement Start	5s
stopTime	Relative Time of Measurement Stop	30s
frameRate	Frame Rate (0Hz = saturated sender)	10Hz
frameSize	Frame Size (0B = flow turned off)	1452B
frameRateString	Outgoing Frame Rate per Stream, separated by “;”	“”
frameSizeString	Outgoing Frame Size per Stream, separated by “;”	“”
maxMsgSize	Maximum Message Size (SCTP and UDP only)	1452B
queueSize	Queue Size (SCTP and TCP only)	1000B
unordered	Fraction of Unordered Messages (SCTP only)	0.0
unreliable	Fraction of Unreliable Messages (SCTP only)	0.0
decoupleSaturatedStreams	Decouple Saturated Streams (SCTP only)	true

Table 5.2: The Parameters of the NETPERFMETER Module

5.5 The NETPERFMETER Simulation Model

The INET FRAMEWORK itself includes some test application modules: *SCTPClient* and *SCTPServer* for SCTP, *TCPBasicClientApp* and *TCPSinkApp* for TCP, and *UDPBasicApp* for UDP. However, these modules lack statistics features and were therefore not really useful for performance analyses. Also, these applications are protocol-specific, i.e. the SCTP application works with SCTP only and its configuration and behaviour significantly differs from the TCP application, etc.. Therefore, trying to compare different protocols with these applications would have been useless. Instead, a new application model has been created from scratch: the NETPERFMETER model. It is also briefly described in [DBPR10b].

The corresponding *NetPerfMeter* simple module has been integrated as a sub-module into the *StandardHost* compound module, as shown in the example of Subfigure 5.1(a). Similar to the real Linux/FreeBSD-based performance metering program NETPERFMETER, which will be introduced in Section 6.3, the application module provides the unidirectional and bidirectional transfer of saturated and non-saturated flows as well as statistics recording. Its parameters are listed in Table 5.2.

For a performance test, a connection is established between two NETPERFMETER instances. This connection establishment process, by using the protocol specified by the parameter protocol (i.e.

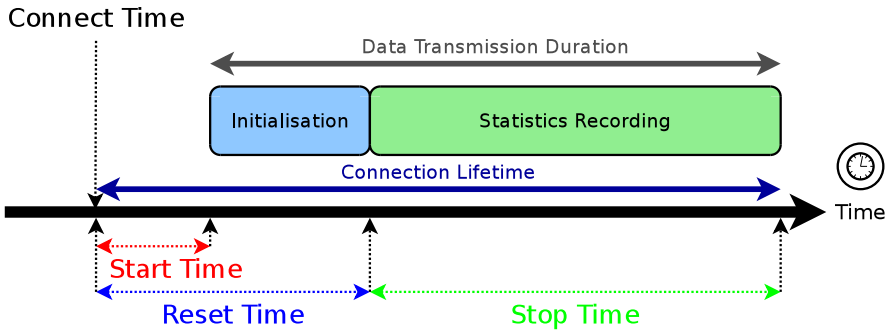


Figure 5.3: An Illustration of the NETPERFMETER Timing Configuration

"SCTP", "TCP" or "UDP"[8]), is triggered by the instance in the so-called *Active Mode* (i.e. the client side). The mode of an instance is configured using the activeMode parameter; a setting of "true" turns the instance into Active Mode. Address and port of the remote instance in the so-called *Passive Mode* (i.e. the server side; configured by setting activeMode to "false") are provided by the parameters remoteAddress and remotePort. The local address and port of an instance – in either mode – may be set by the parameters localAddress and localPort. Since SCTP supports multi-homing, the local address may actually be a list of addresses, or – in the usual case – just left empty. In this case, all available Network Layer addresses are used. The primary path may be set by the primaryPath parameter. Also, for SCTP, the number of outbound streams (parameter: outboundStreams) and the maximum number of inbound streams (parameter: maxInboundStreams) can be configured. These settings are used during the SCTP association setup to negotiate the number of streams in each transfer direction (see also Section 3.5).

Figure 5.3 provides an illustration of the NETPERFMETER connection timing configuration. The connection setup is started at the given *Connect Time* (parameter: connectTime). After a given *Start Time* (parameter: startTime; relative to the begin of the connection establishment), the transfer of payload data starts. At that time, the connection must have been established; otherwise, the simulation stops with an appropriate error message. Usually, the beginning of a communication leads to some kind of irregular initialisation behaviour. For example, the congestion window may have to grow using slow start (see Subsection 2.11.3), etc.. In order to avoid these effects distorting the results, the *Reset Time* time (parameter: resetTime; relative to the connect time) defines the length of a settling time span. After that time, all previously generated statistics are reset. The actual duration of the statistics recording phase is given by the *Stop Time* time (parameter: stopTime; relative to the time of the statistics reset). At the end of this phase, the data transfer is stopped, scalar statistics are written and the connection is finally shut down.

Outgoing payload data is transmitted as frames in a given interval with a given size. Frames are split up into to datagrams, with a maximum size given by maxMsgSize. The parameters to configure frame rate and frame size are frameRate and frameSize, respectively. The special setting of frameSize=0 B turns the flow off; frameRate=0 Hz configures a saturated sender. A saturated sender tries to send as much data as possible. The message queue is therefore filled with up to queueSize messages. Since UDP has no flow control, as described in Subsection 2.13.3, a saturated sender is only possible for SCTP and TCP.

For using SCTP, if there are multiple outbound streams, the frame rate and frame size of each

[8]There is no module for DCCP – see Subsection 2.13.3 – in the INET FRAMEWORK, yet.

stream may be configured separately by providing them as colon-separated strings by the parameters frameRateString and frameSizeString, respectively. The application of strings in this case is necessary, since OMNET++ does not support parameter arrays. Furthermore, it is possible to send a given fraction of the datagrams using unordered delivery (see Section 3.6; parameter: unordered) or using partially reliable transfer (see Subsection 3.11.3; parameter: unreliable). For each message, its kind of delivery (ordered or unordered delivery) and reliability (reliable or partially reliable transfer) is selected randomly, using uniform distribution, according to the configured fractions.

Finally, the parameter decoupleSaturatedStreams turns the decoupling of saturated streams on (i.e. "true", the default) or off (i.e. "false"), as will be further explained in Subsection 7.10.2.

5.6 The Multi-Homed Auto-Routing Module

The configuration of interfaces and routing – by writing a configuration file for each node in the network – is error-prone and rather time-consuming. In order to simplify this task, the INET FRAMEWORK provides modules for so-called *Auto-Routing*. That is, such a module automatically assigns IP addresses to the interfaces and sets up the routing tables of the nodes (e.g. the instances of *StandardHost*). Two of such auto-routing modules have already been provided by the INET FRAMEWORK: *FlatNetworkConfigurator* (for flat, i.e. non-hierarchical network topologies) and *NetworkConfigurator* (for arbitrary topologies). Both apply *Dijkstra's Algorithm* (see [KR08, Subsection 4.2.1]) to build routing tables, according to the least-cost trail among nodes. However, these auto-routing modules do not take multi-homing into account: depending on the configuration, a Path #1 which is intentionally longer (i.e. more expensive) would not be used in favour of a shorter (i.e. less expensive) Path #2; see also Figure 4.2 in Subsection 4.2.2 for an example.

Since auto-routing is highly crucial for a quick and easy simulation configuration, a new auto-routing module for multi-homed networks has been developed from scratch: the *MultihomedFlatNetworkConfigurator* module. It is also briefly described in [DBPR10b]. For each link between two nodes, the new channel parameter netID may be specified. The netID provides the unique identification of an independent network. A network interface connected to a link belongs to the corresponding network. For the routing table computation, Dijkstra's Algorithm is applied on the network corresponding to each netID setting *separately*. That is, paths within one network will not interfere with paths in a different network. The setting of netID=0 has a special meaning: links using this netID value belong to all networks. That is, such links may be shared by multiple networks.

The NED file of a 16-endpoint dual-homed example setup is presented in Listing 3, the resulting setup is shown in Figure 5.4. This scenario is used in an evaluation of CMT-SCTP for MPI (Message Passing Interface) applications in [PTIW07]. The test network consists of a configurable number of endpoints (parameter: numHosts; see lines 27 to 30), connected to a configurable number of independent networks (parameter: numNetworks). Each network has its own router (see lines 31 to 34), which constitutes the centre of a star topology. The channel for the links (defined in lines 18 to 25) defines the attribute netID (in line 23), which is used in lines 39 to 42 to define the connections of the independent networks. Line 5 imports the new *MultihomedFlatNetworkConfigurator* module; it is instantiated in lines 35 to 38. Similar to the auto-routing modules already available in the INET FRAMEWORK, its instance does not require any connections. The @display commands set so-called *Display Strings*, which define how the module instances are displayed on screen (e.g. icons, positioning, background image, etc.; see [Var10, Section 7.1] for details).

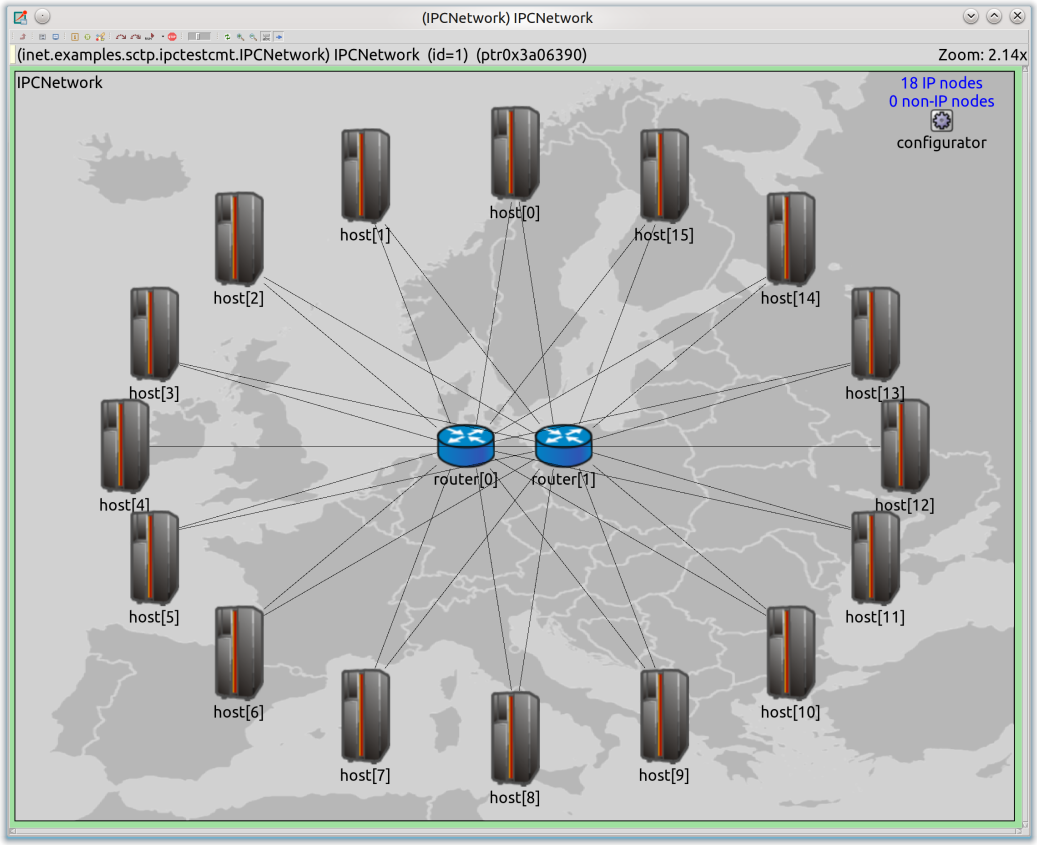


Figure 5.4: A Dual-Homed Example Network

Listing 3 The NED File for the Dual-Homed Example Network

```
package inet.examples.sctp.ipctestcmt;

import inet.nodes.inet.DumpRouter;
import inet.nodes.inet.StandardHost;
import inet.networklayer.autorouting.MultihomedFlatNetworkConfigurator;
import ned.DatarateChannel;

network IPCNetwork
{
    parameters:
        @display("bgi=maps/europe,s;bgb=1024,768");

        int numHosts;
        int numNetworks;
        ...

    types:
        channel path extends DatarateChannel
        {
            // netID is a setting for MultihomedFlatNetworkConfigurator:
            // a trail either belongs to all networks (netID=0) or
            // to a specific network (netID=n, n>0).
            int netID = default(0);
            datarate  = 1Gbps;
        }
    submodules:
        host[numHosts]: StandardHost {
            parameters:
                @display("p=112,84,ring,400,300;i=device/mainframe_vl");
        }
        router[numNetworks]: DumpRouter {
            parameters:
                @display("p=462,384,r,100;i=abstract/router_l");
        }
        configurator: MultihomedFlatNetworkConfigurator {
            parameters:
                @display("p=950,50");
        }
    connections:
        for j=0..numNetworks-1, for i= 0..numHosts-1 {
            host[i].pppg++ <--> path { netID=j+1; } <--> router[j].pppg++;
        }
}
```

5.7 The Simulation Processing Tool-Chain

The extended simulation model – including CMT-SCTP, NETPERFMETER and the *MultihomedFlatNetworkConfigurator* module – have been used for quantitative performance analyses. But such simulations may consist of thousands of individual runs, making parametrisation, processing as well as finally visualising the results a very complex and time-consuming task. Therefore, the SIMPROCTC simulation processing tool-chain has been developed in order to support the following tasks:

- Parametrisation of simulations,
- Distributing the simulation runs in an RSerPool-based computation pool,
- Collecting and post-processing the results as well as
- Visualising the results.

Since the distribution of runs applies the SCTP-based RSerPool framework, this has also been a stress test for SCTP. Particularly, these tests have also delivered some ideas to further enhance the performance of SCTP. Since the details of the tool-chain would exceed the space for this chapter, they can be be found in Appendix B. Furthermore, details on the SIMPROCTC tool-chain have also been published in [DZR09].

5.8 Summary

In this chapter, the simulation environment for the CMT-SCTP evaluation has been introduced: the discrete event simulation package OMNET++ with its INET FRAMEWORK, the CMT-SCTP model itself, the NETPERFMETER application model as well as the auto-routing module for multi-homed networks.

Chapter 6

The Testbed Environment

This chapter describes the setup of the testbed, which has been configured in order to validate the simulation results in reality. Furthermore, the network performance test application NETPERFMETER, which has been designed for the evaluations in the testbed setup, is introduced. In particular, this chapter also shows some lessons learned from building up a distributed network testbed.

6.1 Introduction

Besides a simulative analysis, as described in Chapter 5, it is also crucial to validate results in reality – particularly, in order to validate the simulation model. Also, real hardware may have special or unexpected properties. Such properties may result in a complex interaction with the various protocol mechanisms used by SCTP, as will be shown in detail in Chapter 7.

In order to realise CMT-SCTP in reality, i.e. in a kernel SCTP implementation, the open source kernel implementations introduced in Section 3.13 (see also [DRS+11]) had been considered:

- The Solaris implementation had not been state-of-the-art. Particularly, it had lacked of extensions and API functionalities defined in [STP+11].
- The Linux implementation had mainly been state-of-the-art, but without support of the NR-SACK and CMT-SCTP extensions.
- FreeBSD kernel SCTP had been state-of-the-art and had already provided NR-SACK (see Subsection 3.11.5) and CMT-SCTP as defined by [IAS06] (see Section 4.2).
- Kernel SCTP for MacOS X and Windows are just slightly adapted versions of the original FreeBSD kernel SCTP. But MacOS X requires special and expensive hardware, while Windows puts constraints on the software.

Clearly, the FreeBSD kernel SCTP implementation had been the appropriate choice for the testbed setup, since the basis – particularly, the existing CMT-SCTP implementation – only had to be extended.

System Control	Functionality	Default Setting
net.inet.sctp.cmt_on_off	CMT and Congestion Control Variant	0 (off)
net.inet.sctp.cmt_use_dac	Enable/Disable Delayed Ack. for CMT	0 (off)
net.inet.sctp.nr_sack_on_off	Enable/Disable NR-SACK Extension	0 (off)
net.inet.sctp.buffer_splitting	Buffer Splitting	0 (none)

Table 6.1: The Important CMT-SCTP System Controls of the FreeBSD Kernel

6.2 The FreeBSD Kernel SCTP Implementation

All recent release versions of FreeBSD include SCTP in their default setup. Furthermore, it is possible to obtain the latest developer release of the SCTP implementation, via anonymous CVS[1] from the developers[2], and install a custom kernel[3]. This developer version has been used for the testbed setup, with support by the FreeBSD SCTP developers.

The CMT-SCTP extension already provided by FreeBSD kernel SCTP is parametrised – as all other settings of the kernel – by using *System Controls*. They can be queried and modified using the command `sysctl`; a detailed description can be found in [Fre11, Section 11.11]. Table 6.1 summarises the important CMT-SCTP system controls. By default, CMT is turned off (i.e. net.inet.sctp.cmt_on_off set to 0). Split fast retransmission (see Subsection 4.2.2) as well as PseudoCumAck combined with Retransmission PseudoCumAck as congestion window update strategy (see Subsection 4.2.3) are always active when using CMT. There is no switch to turn them off, since this would make no sense in practise; see also [IAS06, DBPR10b]. The NR-SACK extension (as described in Subsection 3.11.5) can be enabled (1) or disabled (0; the default setting) by the system control net.inet.sctp.nr_sack_on_off.

As part of the DFG project this habilitation thesis has been performed within, further features have been added. Their important *system controls* are shortly introduced here to complete the description. However, the actual description of their *functionalities* follows in Chapter 7 and Chapter 8:

- The NR-SACK extension has been debugged and reworked, since a couple of problems had been found in the original version.
- net.inet.sctp.buffer_splitting configures the variant of buffer splitting (0 – none, 1 – sender-side only, 2 – receiver-side only, 3 – both sides), which will be introduced in Section 7.6. If buffer splitting is enabled, it is always based on outstanding bytes.
- Furthermore, additional settings for net.inet.sctp.cmt_on_off[4] – defining different variants of CMT congestion control – are defined:
  - 0 deactivates CMT (default setting). All other settings turn CMT on.
  - 1 sets plain CMT-SCTP congestion control (as described in Subsection 4.2.1).
  - 2 sets CMT/RPv1 congestion control (to be defined in Subsubsection 8.3.1.1).

[1] Concurrent Versions System; see also [Ced08].

[2] `cvs -d :ext:anoncvs@stewart.chicago.il.us:/usr/sctpCVS co KERN`; the password is "sctp".

[3] See https://nplab.fh-muenster.de/groups/wiki/wiki/4b4b6/Updating_SCTP_sources_on_FreeBSD_82.html.

[4] The name containing "cmt_on_off" may be misleading, but has been retained for backwards compatibility. Originally, the system control had only switched between non-CMT (0) and plain CMT (1) usage.

Feature	IPERF	NETPERF	NTTCP	UPERF	TSCTP	NETPERFMETER
UDP	yes	yes	yes	yes	no	yes
DCCP	no	yes	no	no	no	yes
TCP	yes	yes	yes	yes	no	yes
SCTP	no	limited	no	limited	yes	yes
Multi-Homing	–	no	–	no	yes	yes
Multi-Streaming	–	no	–	no	yes	yes
Unordered Delivery	–	no	–	no	yes	yes
Partial Reliability	–	no	–	no	yes	yes
CMT-SCTP	–	no	–	no	yes	yes
Multiple Flows	no	no	no	yes	no	yes
Bidirectional Flows	no	yes	no	yes	no	yes
Remote Control	no	yes	no	yes	no	yes

Table 6.2: An Overview of Network Performance Test Software

- 3 sets CMT/RPv2 congestion control (to be defined in Subsubsection 8.3.1.2).
- 4 sets MPTCP-like congestion control (to be defined in Subsection 8.3.2).

6.3 The NETPERFMETER Application

In order to evaluate CMT-SCTP setups and to compare them to other protocols – particularly to TCP – in the SCTP testbed which will be described in Section 6.5, the application of existing network test software has been considered first.

6.3.1 Existing Performance Test Software

Table 6.2 summarises the features of the considered application packages IPERF[5], NETPERF[6], NTTCP[7], UPERF[8] and TSCTP[9]. While all examined test applications support FreeBSD, Linux, and MacOS X (i.e. the operating systems being relevant for the testbed), a support for the SCTP protocol is only provided by NETPERF, UPERF and TSCTP. However, only the TSCTP tool – which is a test application provided by the FreeBSD kernel SCTP developers – supports the full capabilities of SCTP: optionally unordered delivery and partially reliable transfer, multi-homing and multi-streaming. However, TSCTP is only applicable for SCTP, which makes a comparison to other protocols – particularly to TCP – impossible.

Besides the support of the full SCTP feature set as well as support for its extensions, the further requirements for the testbed setup have been the possibility to simultaneously set up multiple flows, to perform bidirectional data transfer as well as a remote control capability. The latter denotes that the parametrisation of a measurement run should have been possible at only *one* side of the communication; the remote test endpoint is controlled by the local one, i.e. the complete test configuration is

[5] IPERF: http://Iperf.sourceforge.net/.
[6] NETPERF: http://www.netperf.org/.
[7] NTTCP: http://www.leo.org/~elmar/nttcp/.
[8] UPERF: http://www.uperf.org/.
[9] TSCTP: available in FreeBSD kernel SCTP CVS repository.

performed at the local instance, which automatically ensures that the remote instance gets configured appropriately.

In summary, no existing tool had provided the functionalities and flexibility needed for the CMT-SCTP evaluation in the planned testbed setup. Therefore, a new, multi-platform and open source tool-chain has been designed for this purpose: NETPERFMETER. A description and evaluation of NETPERFMETER has also been published in [DBAR11a].

6.3.2 Design Goals and Features

The key goal of NETPERFMETER is to provide a tool for the performance comparison of multiple transport connections, which are further denoted as *Flows*. That is, it is possible to configure different flows between two systems using varying parameters, in order run a configured measurement, collect the obtained results and post-process them for statistical analyses. Particularly, all four relevant IETF Transport Layer protocols are supported:

1. UDP (see Subsection 2.13.3 and [Pos80]),
2. DCCP (see Subsection 2.13.3 and [KHF06]),
3. TCP (see Subsection 2.13.3 and [Pos81c]), as well as
4. SCTP (see Chapter 3 and [Ste07]).

Of course, this support includes the possibility to parametrise various protocol-specific options, particularly for CMT-SCTP (see Chapter 4). Note, that the protocol support by NETPERFMETER depends on the underlying operating system. DCCP and some SCTP extensions are not available on all platforms, yet.

Furthermore, each flow is able to apply its specific traffic behaviour:

- Each flow may use its own Transport Layer protocol (i.e. UDP, DCCP, TCP or SCTP).
- Bidirectional data transfer is possible.
- Flows may either be saturated (i.e. try to send as much as possible) or non-saturated. In the latter case, a frame rate and a frame size have to be configured. Both may be distributed randomly, using a certain distribution (like uniform, negative exponential, etc.). This feature allows to mimic multimedia traffic (see also [Dre01] for details).
- For the stream-oriented SCTP, an independent traffic configuration is possible for each stream.
- Support for on-off traffic is provided by allowing to specify a sequence of time stamps when to start, stop and restart a flow or stream.
- Also, for SCTP, it is possible to configure partial reliability (see Subsection 3.11.3) as well as ordered and unordered delivery (see Section 3.6).

Clearly, the NETPERFMETER application provides features similar to the NETPERFMETER simulation model introduced in Section 5.5. It is therefore relatively easy – from the parametrisation perspective – to reproduce NETPERFMETER simulation scenarios in reality.

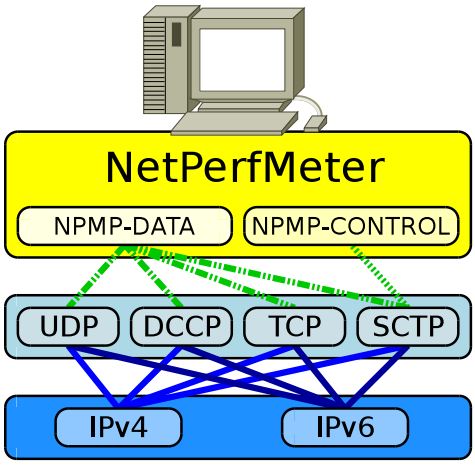


Figure 6.1: The NetPerfMeter Protocol Stack

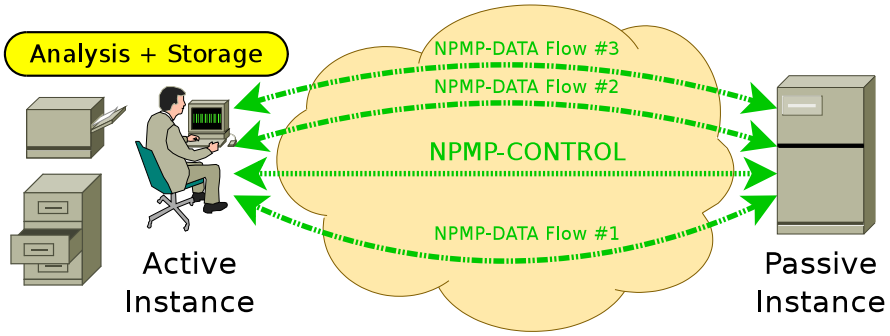


Figure 6.2: The Concept of a NetPerfMeter Measurement

6.3.3 Instances and Protocols

Similar to the NETPERFMETER simulation model described in Section 5.5, an application instance may either be in *Active Mode* or *Passive Mode*. Figure 6.1 illustrates the protocol stack of a NETPERFMETER node; the concept of a measurement is presented in Figure 6.2. The passive instance accepts incoming NETPERFMETER connections from the active instance. The active instance controls the passive instance, by using a control protocol denoted as NETPERFMETER Control Protocol (NPMP-CONTROL). That is, the passive instance may run as a daemon; no manual interaction by the user – e.g. to restart it before a new measurement run – is required. This feature is highly practical for a setup distributed over multiple Internet sites (like the distributed SCTP testbed to be described in Subsection 6.5.2) and allows for parameter studies consisting of many measurement runs. The payload data between active and passive instances is transported using the NETPERFMETER Data Protocol (NPMP-DATA).

6.3.4 Measurement Processing

Figure 6.3 presents the message sequence of a NETPERFMETER measurement run.

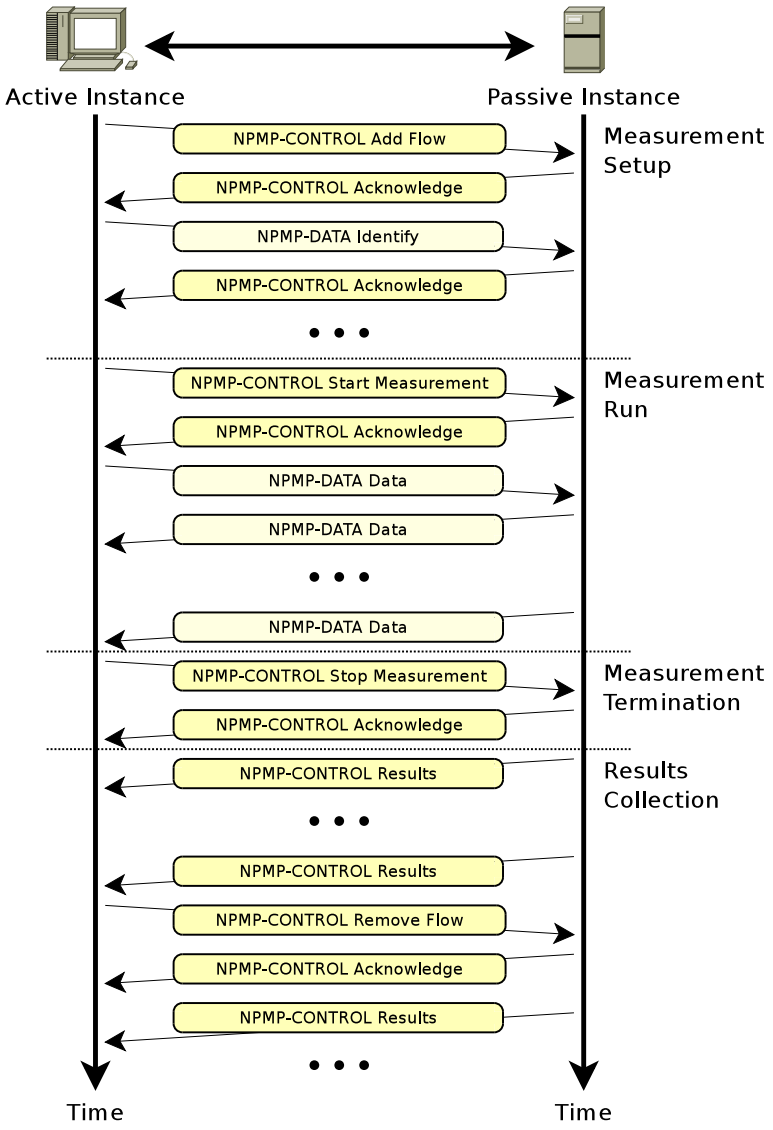


Figure 6.3: A Measurement Run with NETPERFMETER

6.3.4.1 Measurement Setup

A new measurement run setup is initiated by the active NETPERFMETER instance by establishing an NPMP-CONTROL association (using SCTP for transport) to the passive instance first. Then, the configured NPMP-DATA connections are established by their configured Transport Layer protocols[10]. The passive NETPERFMETER instance is informed about the identification and parameters of each new flow by using NPMP-CONTROL Add Flow messages. On startup of the NPMP-DATA flow, an NPMP-DATA Identify message allows the mapping of a newly incoming connection to a configured flow by the passive instance. It acknowledges each newly set up flow by an NPMP-CONTROL Acknowledge message. After setting up all flows, the scenario is ready to start the measurement run.

[10]For the connection-less UDP, the message transfer is just started.

6.3.4.2 Measurement Run

The actual measurement run is initiated from the active NETPERFMETER instance using an NPMP-CONTROL Start Measurement message, which is also acknowledged by an NPMP-CONTROL Acknowledge message. Then, both instances start running the configured scenario by transmitting NPMP-DATA Data messages over their configured flows.

During the measurement run, incoming and outgoing flow bandwidths may be recorded as vectors – i.e. time series (see also Section 5.2) – at both instances, since NPMP-DATA Data traffic may be bidirectional. Furthermore, the CPU utilisations – separately for each CPU and CPU core – are also tracked. This allows to identify performance bottlenecks, which is particularly useful when debugging and comparing transport protocol implementation performance. Furthermore, the one-way delay of messages can be recorded. Of course, in order to use this feature, the clocks of both nodes need to be appropriately synchronised, e.g. by using NTP (see Subsection 2.13.4).

6.3.4.3 Measurement Termination

The end of a measurement run is initiated – from the active NETPERFMETER instance – by using an NPMP-CONTROL Stop Measurement message. Again, it is acknowledged by an NPMP-CONTROL Acknowledge message. At the end of the measurement, average bandwidth and one-way delay of each flow and stream are recorded as scalars (i.e. single values; see also Section 5.2). They may provide an overview of the long-term system performance.

6.3.5 Result Collection

After stopping the measurement, the passive NETPERFMETER instance sends its global vector and scalar results (i.e. over all flows) to the active instance, by using one or more NPMP-CONTROL Results messages. Then, the active NETPERFMETER instance sequentially removes the flows by using NPMP-CONTROL Remove Flow messages, which are acknowledged by NPMP-CONTROL Acknowledge messages. On flow removal, the passive instance sends its per-flow results for the corresponding flow, again by using NPMP-CONTROL Results messages.

The active instance, as well, archives its local vector and scalar results data and stores them – together with the results received from its peer – locally. All result data is compressed by using BZIP2 (see [Sew07]), which may save a significant amount of bandwidth[11] and disk space.

6.3.6 Measurement Execution, Result Post-Processing and Visualisation

By using shell scripts, it is possible to apply NETPERFMETER for parameter studies, i.e. to create a set of runs for each input parameter combination. For example, a script could iterate over a send buffer size σ from 64 KiB to 192 KiB in steps of 64 KiB as well as a path bandwidth ρ_X from 10 Mbit/s to 100 Mbit/s in steps of 10 Mbit/s and perform 5 measurement runs for each parameter combination.

The result post-processing and visualisation step is identical to the procedure applied for the simulations. The details can be found in Section B.4.

[11] Of course, the passive node compresses the data *before* transfer.

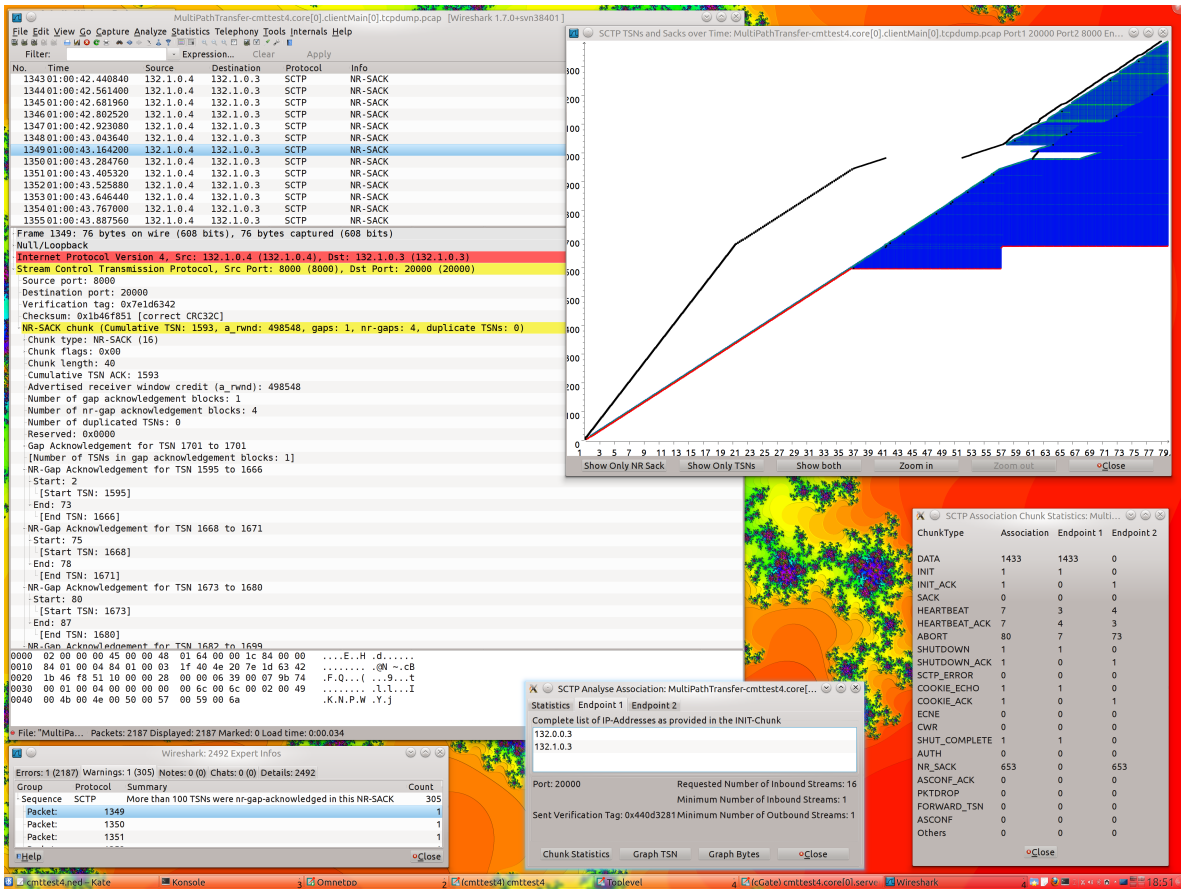


Figure 6.4: An SCTP Packet Trace Analysis with WIRESHARK

6.3.7 Reusability

NETPERFMETER has been designed with reusability in mind. Therefore, it has been released as open source under GPLv3 licence. It is freely available for download on its web site[12]. Furthermore, it has been contributed to Debian and Ubuntu Linux (allowing to install it directly from the distributions' standard package repositories) as well as to FreeBSD (allowing to install it from the FreeBSD ports collection). MacOS X and Solaris are also supported. Furthermore, the IANA has assigned SCTP PPIDs (see also Section 3.6) and a DCCP Service Code (see [KHF06, Subsection 19.8]) for the NETPERFMETER protocols.

6.4 Wireshark and the SCTP Analysis Tools

A very helpful tool during the test and debugging work with both, FreeBSD kernel SCTP as well as the simulation model described in Chapter 5, has been WIRESHARK[13]. It is an open source packet trace capturing and analysis tool; a very detailed introduction is provided by [LSW12].

[12] NETPERFMETER: http://www.iem.uni-due.de/~dreibh/netperfmeter/.
[13] WIRESHARK: http://www.wireshark.org/.

WIRESHARK provides so-called *Dissectors* to decode packets, which may either be captured in real-time from a network interface or read from a PCAP file. Figure 6.4 provides a screenshot of an SCTP packet trace analysis session with WIRESHARK: the main window (upper left) provides a view of the packet trace (upper part) and the decoded fields of a selected packet (lower part; here: an SCTP packet containing an NR-SACK chunk). Besides the plain decoding feature, WIRESHARK also offers SCTP association analysis tools:

- The *Chunk Statistics* tools (in the lower middle and right windows) provide a summary of the transferred chunk types and the amount of data sent in each transport direction. They give a brief overview whether the SCTP communication is working as expected.
- Furthermore, the *TSN Graph* tool (in the upper right window) provides a graphical representation of the sent and acknowledged (CumAck'ed and GapAck'ed; see also Section 3.7) TSNs. This tool has been highly useful during the validation phase of the simulation model as well as for analysing performance issues on dissimilar paths (to be explained in Chapter 7).
- Also, the *Expert Info* tool (in the lower left window) provides an analysis for potential misbehaviour of the SCTP communication. For example, it validates TSN sequences, checks for potential protocol violations, "unusual behaviour", etc.. In particular, this tool has been extended – as part of this work – with a validity check of GapAck TSN values.

For testing and debugging, WIRESHARK has also been equipped with dissectors for the two NETPERFMETER protocols (i.e. NPMP-CONTROL and NPMP-DATA; see Section 6.3) and the Scripting Service Protocol (to be introduced in Subsection B.3.5). Both have been applied for stress tests. Furthermore, a dissector for the Component Status Protocol (to be introduced in Subsection B.3.4) has been added. All dissectors have been contributed to the WIRESHARK project and are now directly provided within its application package.

6.5 The Testbed

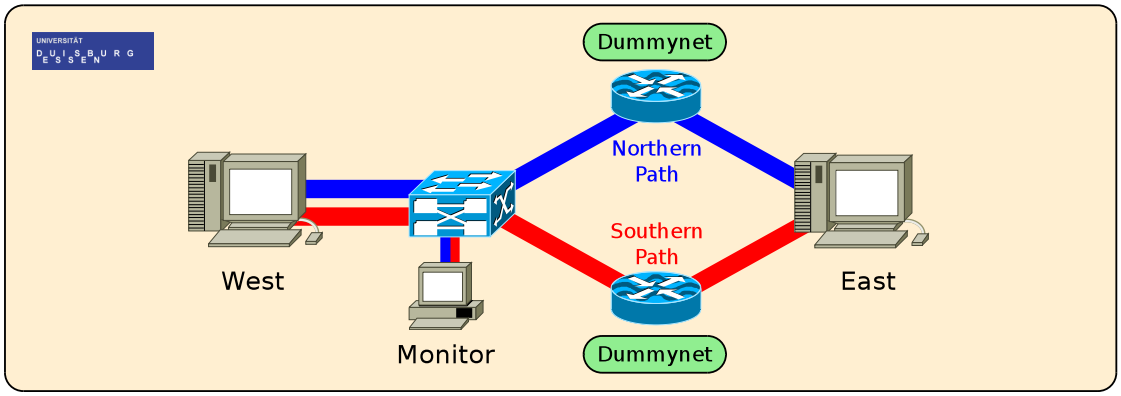
Figure 6.5 presents the two setups of the testbed: a local setup (see Subfigure 6.5(a)) as well as a distributed Internet setup between two sites (see Subfigure 6.5(b)). For both setups, PCs with dual-core AMD CPUs under 64-bit FreeBSD 8.2[14] with the development version of SCTP (see Section 6.2) are used. The PCs used for the protocol tests are equipped with Intel 82576-based, server-quality, dual-port Gigabit Ethernet NICs providing support for SCTP CRC-32C checksum offloading (see Subsection 3.12.3). NETPERFMETER version 1.1.8 (see Section 6.3) as well as WIRESHARK (see Section 6.4) are installed on all nodes. Furthermore, all nodes run NTP (see Subsection 2.13.4) to ensure accurately synchronised clocks, as well as SSH (see Subsection 2.13.4) to provide remote logins.

6.5.1 Local Setup

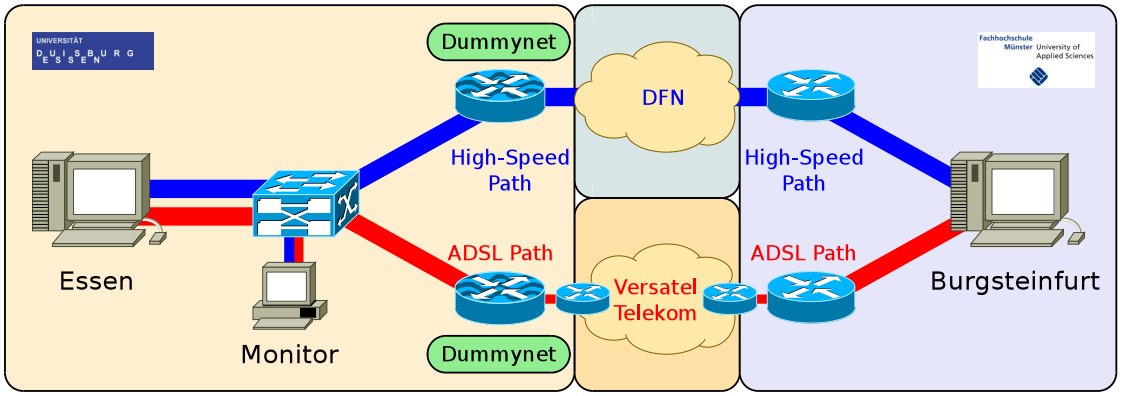
The local testbed setup illustrated in Subfigure 6.5(a) uses the two PCs "West" and "East" as communications endpoints. The routers on the two paths, "Northern Path" (in blue colour) and "Southern Path" (in red colour), apply DUMMYNET to provide certain QoS characteristics: bandwidth limitation, delay, bit errors and packet losses. DUMMYNET is part of the packet filtering infrastructure in the FreeBSD kernel[15]. Further details are provided by [CR09].

[14] FreeBSD: http://www.freebsd.org/.

[15] DUMMYNET is an optional part of the FreeBSD kernel and must be activated at time of kernel compilation.



(a) Local Testbed in the Networking Lab in Essen



(b) Distributed Testbed between Essen and Burgsteinfurt

Figure 6.5: An Illustration of the Testbed Setup

In order to simplify the capture of packet traces on the two paths, the PC "Monitor" is connected to the monitoring port of a managed switch; it is therefore able to see the packets of both paths and record their actual timing sequence. This hardware-based solution for live packet capturing and analysis is necessary, since the FreeBSD version of WIRESHARK is limited to capturing packets from only one network interface (i.e. of a single path) at a time. The only alternative to obtain a multi-interface packet trace on FreeBSD is to capture on each interface separately, and finally merge the obtained PCAP trace files. Of course, this approach would prevent any real-time analysis of a multi-homed communication.

6.5.2 Distributed Setup

In order to perform real Internet measurements, the testbed has been extended by an Internet setup, as depicted in Subfigure 6.5(b). It links the sites of the two main project partners (see also Section 1.2):

- the Computer Networking Technology Group at the Institute for Experimental Mathematics of the University of Duisburg-Essen in Essen/Germany[16] and
- the Department of Electrical Engineering and Computer Science at the Münster University of Applied Sciences in Burgsteinfurt/Germany[17].

Both sites are interconnected via two disjoint paths:

1. The first path (which is shown in blue colour) uses a high-speed fibre optic connection over the German Research Network (Deutsches Forschungsnetz[18] – DFN), with a typical RTT of 4 ms.
2. The second path (which is shown in red colour) uses an ADSL connection in Essen (Versatel[19]; 800 Kbit/s upstream, 16 Mbit/s downstream) as well as an ADSL connection in Burgsteinfurt (Telekom[20]; 1 Mbit/s upstream, 16 Mbit/s downstream); the typical RTT is 56 ms.

That is, the QoS characteristics of these two paths are highly dissimilar. As will be shown in Chapter 7, this is an interesting challenge for CMT.

Like for the local setup described in Subsection 6.5.1, the distributed scenario may also use DUMMYNET for further adaptation of the QoS characteristics. Furthermore, the PC "Monitor" is again able to perform a real-time analysis of the traffic on both paths, by using WIRESHARK.

6.5.3 The Reality – Challenges and Lessons Learned

While the schematic setup of the testbed illustrated in Figure 6.5 is straightforward and reasonably simple, the reality is somewhat more complicated and challenging. Figure 6.6 presents the actual setup at the site in Essen. The two PCs on the left-hand side on the table are the two routers; the two PCs on the left-hand side below the table are the endpoints; the monitoring PC is located on the table at the right-hand side of the display. When setting up this testbed environment, a couple of challenges had to be solved. These challenges and their solutions are shortly described here, in order to provide some "lessons learned" to other researchers who are going to build up similar setups.

[16] Location of the endpoint in Essen: 51° 28' 32.16" North, 7° 0' 42.27" East.
[17] Location of the endpoint in Burgsteinfurt: 52° 8' 53.94" North, 7° 20' 1.69" East.
[18] Deutsches Forschungsnetz: http://www.dfn.de/.
[19] Versatel: http://www.versatel.de/.
[20] Telekom: http://www.telekom.de/.

(a) A Complete Picture (b) Routers, Ethernet Switches, KVM Switch and PCU

Figure 6.6: The Testbed Setup in Essen in Reality

6.5.3.1 Power Control Unit and Keyboard/Video/Mouse Switch

First, since the testbed is also used for debugging the SCTP implementations, system crashes and deadlocks may occur. This frequently happens at very inappropriate times, e.g. when the building is closed due to week-ends or public holidays. In order to avoid the need of physical access to the PCs in order to reset them, all systems are attached to an Internet-connected *Power Control Unit* (PCU). It provides a web interface for turning the power to selected nodes on or off, i.e. providing the possibility for a remote-controlled hard reset.

Furthermore, in order to reduce hardware requirements, all PCs are connected to a so-called *Keyboard/Video/Mouse* (KVM) switch: it allows to share a single keyboard, mouse and video display among all PCs.

6.5.3.2 Peculiarities of DUMMYNET

While DUMMYNET – as introduced by [CR09] – is a well-known and widely used tool to provide certain QoS characteristics, it is also important to know its peculiarities. The limitations of DUMMYNET have been examined in some further detail in [BDRF11]. Particularly notable are the following points:

- The emulation of jitter is very limited. This restriction is important for real-time multimedia traffic (e.g. telephony; see also [Ste00]). Since this kind of traffic is not used within the context of this thesis, the limited jitter support is not important here. However, since CMT for multimedia traffic is a topic of future work, it is important to mention here.

- Queue sizes are limited to 100 packets at most. This prevents the emulation of very long queues, which are realistic for ADSL connections. However, since the testbed uses a *real* ADSL connection, this limitation is not problematic in the context of this thesis.

An alternative to DUMMYNET may be NETEM[21], a QDisc for applying QoS characteristics, which is provided by the traffic control infrastructure of the Linux kernel and described by [Hem05]. Of

[21] NETEM: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

course, its application would mean to replace the FreeBSD-based routers by Linux-based routers. [BDRF11] provides a detailed performance comparison between DUMMYNET and NETEM.

6.5.3.3 Challenges of ADSL Configuration

The testbed – as illustrated in Figure 6.5 – consists of multiple subnets. In case of the DFN network (in blue colour), the necessary subnetting – see also [Tan96] – has been realised without any problem. However, performing a subnet configuration for the DSL path (in red colour) had been more complicated. The ISP had delivered a preconfigured ADSL router, providing an Ethernet port with one large network configured. For subnetting, of course, static routes would have to be configured on this device. However, its configuration had been fixed by the ISP – with no possibility to reconfigure it[22]. Therefore, the solution had been to apply Proxy ARP (defined in [CMQ87] as RFC 1027) on the FreeBSD router, in order to pretend one large network to the ADSL router and apply all static route configuration on the own FreeBSD machine. For the communication performance, this setup has no significant influence – its configuration is just awkward.

One more restriction caused by the ADSL network has been the lack of support for IPv6. Native IPv6 connectivity is still very unusual for end-users in Germany. Most of the ISPs – including the ADSL ISPs used for the testbed – still do not support it, yet. With IPv6 connectivity only via the DFN network, the measurements had therefore been restricted to IPv4 only. However, the general behaviour to be examined in this thesis is independent of the underlying Network Layer protocol. Therefore, IPv6 connectivity would just have been a "nice to have" feature for implementation testing.

6.5.3.4 Challenges of Ethernet Hardware

A further important insight of the testbed setup is that Ethernet $\neq$ Ethernet. Originally, in order to set up the two paths illustrated in Figure 6.5, the on-board Ethernet interface as well as an additional Ethernet NIC had been used for the PCs (i.e. endpoints and routers). However, this had led to initially inexplicable bandwidth drops during performance tests, i.e. the on-wire speed of 1 Gbit/s had irregularly been reduced – for the fraction of a second – to 100 Mbit/s, and returned back to 1 Gbit/s afterwards. An intensive testing to find the problem, which – in combination with some bugs in the kernel – had taken several weeks, has finally identified the interaction of multiple problem sources:

- It had been shown that some combinations of cables, switches and NICs did not work well together. Therefore, all cables had finally been replaced by straight from the factory ones of the same vendor.

- The switches – which had initially been medium-cost workgroup switches – had been replaced by more expensive devices of a well-known network equipment vendor.

- Some further tests had also shown some different behaviour of the NICs. Particularly, different NIC vendors realise different queue lengths and characteristics. These differences, although very small, had led to slightly dissimilar paths. However, these small dissimilarities had already been sufficient to have a significant impact on the CMT-SCTP performance, as will be shown in Chapter 7. Finally, Intel 82576-based, server-quality, dual-port NICs (i.e. one port for each path) had been installed in all PCs of the setup.

[22] Applying a password recovery procedure would have been a solution for this problem, but – of course – this would have violated the ADSL contract with the ISP.

The described changes have finally resulted in the testbed to behave as expected. But, at least, there is also a positive effect of the prolonged search for the problem sources: multiple bugs in the FreeBSD kernel SCTP implementation had been identified and fixed and – most importantly – the tests have pointed out to the interesting research topic of CMT performance on dissimilar paths.

6.6 Virtualisation of the Testbed

Due to the various difficulties experienced when setting up the testbed, a partial virtualisation is considered for the future continuation of the research project on SCTP. Particularly, it is intended to utilise the research platform G-LAB[23] (German Lab), which is introduced in detail by [TGFS+09], for this task. The G-LAB consists of a group of nodes, which are available as a large testbed for computer networking and distributed systems research. A research project has access to a so-called *Slice*, which consists of virtual machine access to a subset of the nodes. Virtual links may be combined with real-existing physical links (like an ADSL connection or a wireless channel) to set up complex topologies and perform experiments in these scenarios. This will bring significant benefits in terms of scalability, extensibility, reliability and manageability of a testbed.

6.7 Summary

In this chapter, the FreeBSD-based, distributed testbed environment for the CMT-SCTP evaluation has been introduced. This environment includes the NETPERFMETER network performance test application, which has been developed in order to efficiently perform measurements in the testbed. Furthermore, the packet trace analysis tool WIRESHARK – which has been used intensively for debugging and testing the setup as well as during validation of the simulation model – has been described briefly. Also, some important lessons learned from building up a distributed network testbed have been shown.

[23] G-LAB: http://www.german-lab.de/.

Chapter 7

Efficient Handling of Dissimilar Paths

This chapter indicates the begin of the CMT-SCTP evaluation part. In this chapter, the performance evaluation of CMT-SCTP over dissimilar paths is described and analysed. First, the performance issues for unordered delivery are analysed and solved by additional mechanisms. Based on these mechanisms for unordered delivery, the evaluation and optimisation is extended to ordered delivery as well. Finally, mechanisms to improve the performance of certain multi-streaming applications are introduced.

7.1 Introduction

CMT-SCTP, as defined by [IAS06] and explained in Section 4.2, looks very straightforward at first sight. However, this approach assumes that the paths used for the intra-flow load sharing are relatively similar, i.e. they provide similar QoS characteristics like bandwidths, delays and bit error rates (see also Section 2.4). As will be shown in this chapter, dissimilarities of these QoS characteristics cause severe performance degradations. These problems are also not restricted to CMT-SCTP; they are generic problems and apply to other Transport Layer approaches – particularly to MPTCP – as well.

Clearly, one of the core application scenarios of CMT-SCTP is bulk data transfer, e.g. in MPI setups as described by [PTIW07]. Therefore, the performance metric is defined by the achievement of the following two goals:

1. The overall payload throughput (see also [MA01]) should be maximised.
2. Furthermore, the overall payload throughput for CMT-SCTP should at least be equal to the throughput of standard SCTP over the best path.

Unlike TCP/MPTCP, SCTP provides support for optional unordered delivery and multi-streaming. That is, the three underlying protocol mechanisms of SCTP can be used and examined separately:

1. Unordered delivery (see Section 3.6),
2. Ordered delivery (see also Section 3.6 as well as Section 2.8) and
3. Multi-streaming (see Section 3.5).

Further details on the separation of these functional blocks can also be found in [DBHR10]. Due to this separation, the resolution of the problems described in this chapter is simplified by applying

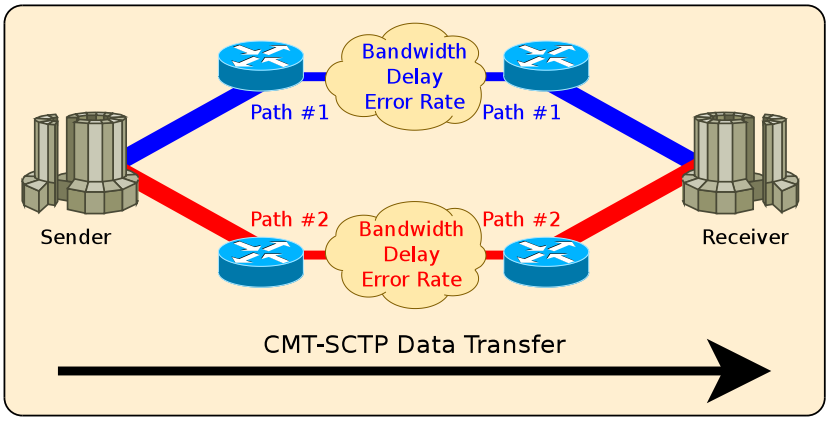


Figure 7.1: The Scenario Setup for the Performance Evaluations

the "Divide and Conquer" principle. That is, at first, the challenges of unordered delivery are examined and solved in Section 7.5, Section 7.6 and Section 7.7. This is followed by ordered delivery in Section 7.8 and Section 7.9. Finally, multi-streaming is analysed in Section 7.10.

7.2 Scenario Setup

For the performance evaluation of CMT-SCTP, the simulation setup introduced in Chapter 5 as well as the testbed setup introduced in Chapter 6 have been used. The general scenario is illustrated in Figure 7.1. Unless otherwise specified, the following configuration parameters have been set:

- The sender has transferred a unidirectional SCTP payload data flow consisting of one stream to the receiver. The stream has been saturated, i.e. the sender has tried to transfer as much data as possible. The message size has been 1,452 bytes at an MTU of 1,500 bytes (i.e. MTU-sized packets, see also [Ste07]).
- SCTP has used the standard settings defined in RFC 4960 (see also Chapter 3).
- SCTP has applied burst mitigation by using the variant "Max Burst" from [AB05] adapted to SCTP. Burst mitigation and the reasons for this choice in favour of "Use It or Lose It" (see Section 3.9) will be explained in detail in Subsection 7.9.4.
- The SCTP extensions SACK immediately (see Subsection 3.11.6) and NR-SACK (see Subsection 3.11.5) have been active, since they are provided by state-of-the-art SCTP implementations (see Section 3.13 and [DRS+11]).
- The SCTP packet drop reporting extension (see Subsection 3.11.8) has been turned off. It will be explicitly enabled in bit error scenarios.
- Of course, the CMT-SCTP extension as described in Section 4.2 has been active. Also, the additional mechanisms defined by [IAS06] have been applied: split fast retransmission (see Subsection 4.2.2), congestion window update for CMT, version 2 (see Subsection 4.2.3) and delayed acknowledgement for CMT (see Subsection 4.2.4).

- The additional mechanisms that have been developed as part of this thesis have been configured as follows: smart SACK path selection (to be introduced in Subsubsection 7.9.4.2), smart fast retransmission (to be introduced in Subsection 7.7.3; published in [DBRT10]) and buffer splitting (both sides, based on outstanding bytes; to be introduced in Section 7.6; published in [DBRT10, ADB+11]) have been activated. Chunk rescheduling (to be introduced in Section 7.8; published in [DBRT10]) has been turned off.

- The QoS characteristics of the two paths (Path #1 shown in blue colour; Path #2 shown in red colour) have been varied. By default, they have used a bandwidth of 100 Mbit/s, a delay of 1 ms (realistic for a LAN setup) and no bit errors. The queue of each router applying the QoS settings is a RED queue (see Subsection 2.4.2 and [FJ93]), using the parameters MinTh=30, MaxTh=90 and MaxP=10%. That is, the configuration has followed the recommendations by [Flo97]. All other queues have been FIFO queues (see Subsection 2.4.2) with a capacity of 100 packets.

- The runtime of each simulation or measurement run has been 60 s. Tests have been repeated multiple times (at least 24 times for simulations, at least 8 times for measurements) in order to ensure a sufficient statistical accuracy. All result plots show the average values over these repetitions and their corresponding 95% confidence intervals (details are described in Subsection B.4.2).

7.3 Model Validation on Similar Paths

A mandatory preliminary work of each simulative performance evaluation is to validate the model. This validation has been performed by comparing the results of testbed measurements and simulation results, for a similar path setup varying the bandwidth ρ of each path from 1 Mbit/s to 100 Mbit/s. This initial performance evaluation has also been published in [DBPR10b]. The resulting application payload throughput results are presented in Figure 7.2 (testbed results in Subfigure 7.2(a), simulation results in Subfigure 7.2(b)).

As parameters for this evaluation, only the features of plain CMT-SCTP as defined by [IAS06] and the defaults of standard SCTP as defined in RFC 4960 (see [Ste07]) have been used:

- Burst mitigation as suggested by [Ste07, Subsection 6.1] has been applied (i.e. the variant "Use It or Lose It" – see Section 3.9).

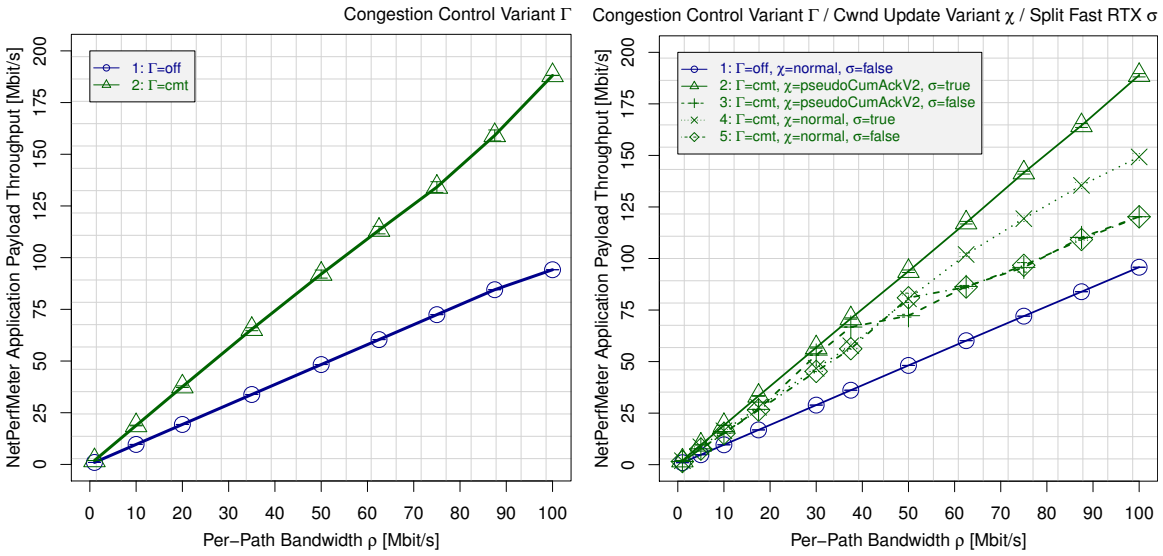
- The SCTP extensions SACK immediately (see Subsection 3.11.6) and NR-SACK (see Subsection 3.11.5) have been turned off.

- Send buffer and receive buffer have had a size of 64 MiB (i.e. they have been *huge*, in order to avoid any lack of space).

The expected "ideal case" throughput in this scenario can be computed theoretically, by using the known sizes of payload and overhead:

- The payload of each SCTP packet is 1,452 bytes.

- The overhead consists of SCTP and DATA chunk headers (28 bytes in total), IPv4 header (20 bytes), Ethernet header (14 bytes) as well as Ethernet preamble (8 bytes) and checksum (4 bytes). That is, the total frame size is 1,526 bytes.



(a) Testbed Measurement Results (b) Simulation Results

Figure 7.2: The CMT-SCTP Performance in a Similar Paths Setup

That is, the application throughput for using one path (i.e. non-CMT) as well as two paths (i.e. CMT) can be calculated as follows:

$$100\ \text{Mbit/s} * \frac{1,452\ \text{B}}{1,526\ \text{B}} \approx 95\ \text{Mbit/s},$$
$$2 * 100\ \text{Mbit/s} * \frac{1,452\ \text{B}}{1,526\ \text{B}} \approx 190\ \text{Mbit/s}.$$

Both, testbed measurement and simulation model reasonably well reach the expected throughput performance for non-CMT at ρ=100 Mbit/s (curve 1; drawn in dark blue colour), while the corresponding CMT performance is around 188 Mbit/s (curve 2; represented by a solid line in dark green colour). This is just about 1% less than the theoretic optimum. Also, as expected, the simulation results show the need for split fast retransmission (curves 2 and 4; see Subsection 4.2.2) and congestion window update for CMT, version 2 (curves 2 and 3; see Subsection 4.2.3). Since turning these options off is – obviously – not useful, a corresponding configuration option is not available in the FreeBSD kernel SCTP implementation. Measurement curves are therefore not available here. In summary, testbed measurements and simulation results reach corresponding results, and the performance of CMT-SCTP in this *nearly* similar setup is reasonably well.

Of course, there is a reason for the about 1% performance loss: the RED queues (see Subsection 2.4.2) introduce some temporary dissimilarities into the system. At non-deterministic times, the RED queue drops packets and the sender has to adapt the congestion window, according to the AIMD behaviour introduced in Section 3.8. These small dissimilarities trigger the issues also occurring on intentionally dissimilar paths, where these problems have a significantly higher impact.

7.4 Buffer Size Considerations

The sizes of send and receive buffer – as explained in Subsection 2.9.2 – are very important system parameters. While the configuration of the setup in Section 7.3 has used extremely large buffers to avoid space shortages, the sizes of these buffers must be limited in practise, in order to fulfil memory constraints of the used systems. It is therefore necessary to carefully configure the buffer sizes.

In an ideal scenario without any losses, the absolute lower bound $B^A_{\min}$ for the size of send and receive buffers, in a setup with paths $P = \{P_1, \ldots, P_n\}$ as well as $\mathrm{Bandwidth}_i$ the bandwidth and RTT_i the RTT of path P_i, is defined by the bandwidth-RTT product as described in Subsubsection 2.9.2.3:

$$B^A_{\min} = \max_{1 \leq i \leq n} \{\mathrm{RTT}_i\} * \sum_{i=1}^{n} \mathrm{Bandwidth}_i.$$

That is, the buffer has to store at least the traffic generated within an RTT on the slowest path. After that time, a CumAck frees space to transmit or receive new DATA chunks.

In reality, the AIMD congestion control behaviour, as introduced in Subsection 2.11.2, leads to frequent losses of single packets – which is an intended behaviour: the congestion control tries to utilise all available bandwidth by increasing the congestion window and gets a feedback from the network – in form of packet losses – when too much data has been sent. These losses are handled by fast retransmissions, which should therefore also be covered by the buffer to avoid transmission interruptions. That is, the minimum buffer size $B^F_{\min}$ for the send and receive buffers is:

$$B^F_{\min} = 2 * \left[\max_{1 \leq i \leq n} \{\mathrm{RTT}_i\} * \sum_{i=1}^{n} \mathrm{Bandwidth}_i \right].$$

Timer-based retransmissions occur in case of high congestion, i.e. they should be rare. In order to also cover a timer-based retransmission, the minimum buffer size $B^T_{\min}$ for the send and receive buffers is:

$$B^T_{\min} = \left(3 * \max_{1 \leq i \leq n} \{\mathrm{RTT}_i\} + \max_{1 \leq i \leq n} \{\mathrm{RTO}_i\} \right) * \sum_{i=1}^{n} \mathrm{Bandwidth}_i.$$

That is, in the worst case it takes three times the highest path RTT (first transmission, fast retransmission, timer-based retransmission) plus the highest path RTO (see Subsection 2.11.4).

Since the default *minimum* RTO is RTO.Min=1 s (as introduced in Section 3.8), the timer-based retransmission coverage by the buffer space is usually too expensive. Figure 7.3 illustrates the buffer space requirements for varying the sum of path bandwidths and highest RTT R for coverage of only the first transmission (Θ=1st; solid lines), a fast retransmission (Θ=Fast, dashed lines) and a timer-based retransmission (Θ=Timer-Based; dotted lines). For example, a 100/100 Mbit/s setup (i.e. a bandwidth sum of 200 Mbit/s) having a highest path RTT of 110 ms (e.g. an inter-continental communication; see also Subsection 2.4.2) requires send and receive buffer spaces of 32 MiB to cover a timer-based retransmission. Considering a system with hundreds or thousands of simultaneous associations (e.g. in MPI setups as described by [PTIW07]), realistic scenarios can just cover a fast retransmission – which still requires a buffer space of about 5.5 MiB here.

As an example, the default buffer size setting of FreeBSD kernel SCTP[1] is only 233,016 bytes. Obviously, buffer space is a limited resource – and this leads to problems.

[1] FreeBSD release 8.2.

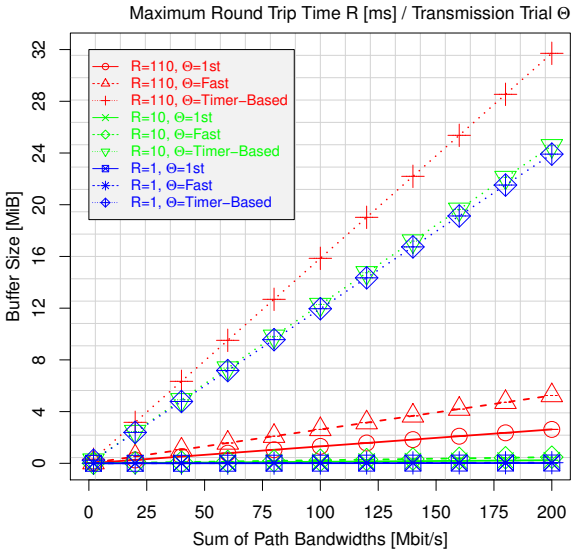


Figure 7.3: Required Sizes for Send and Receive Buffers

7.5 Buffer Blocking Issues

The core problem occurring when transmitting over dissimilar paths is caused by send and receive buffer limitations, i.e. the problem subdivides between sender side and receiver side. Furthermore, for each side, two sub-problems have been identified as part of this thesis. The initial results on these so-called *Buffer Blocking* issues have been published in [DBRT10], with some refinements and the classification in [ADB+11].

7.5.1 Send Buffer Blocking

Sender-side buffer blocking is denoted as *Send Buffer Blocking*. It can be classified into two sub-problems.

7.5.1.1 Transmission-Induced Send Buffer Blocking

The first sender-side buffer blocking issue is denoted as *Transmission-Induced Send Buffer Blocking*. An example is depicted in Figure 7.4. In this example, Endpoint A transmits DATA chunks over two paths to Endpoint B. The send buffer of Endpoint A has the capacity to store eight DATA chunks; it has been completely filled with outgoing data. The sender has transmitted seven DATA chunks over the high-delay Path #1 (TSNs #19 to #25) and one over the low-delay Path #2 (TSN #18), since the congestion windows allow[2] seven and one DATA chunks, respectively. This, of course, is exactly the intended behaviour of the congestion control (see Section 3.8 and Subsection 2.11.2).

The problem arises when TSN #18 – which has been sent over the low-delay path – gets acknowledged to the sender side by the reception of a SACK chunk. According to the rules for increasing the congestion window on reception of a new acknowledgement, the congestion window on Path #1 may grow (see Subsubsection 2.11.2.1 and Subsection 4.2.3). However, since only DATA chunk #18 leaves

[2]For simplification, the congestion windows are given in units of DATA chunks here. In reality, they are given in units of bytes. Nevertheless, the principle is the same.

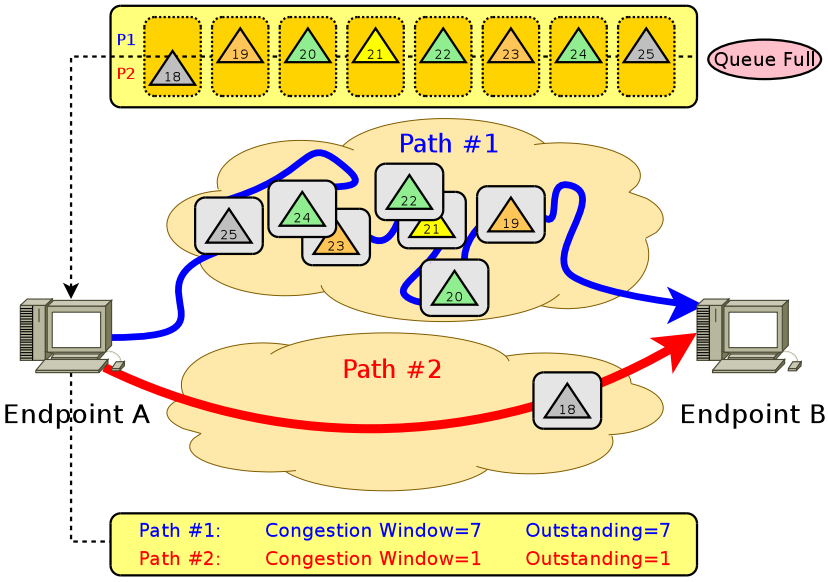


Figure 7.4: An Example for Transmission-Induced Send Buffer Blocking

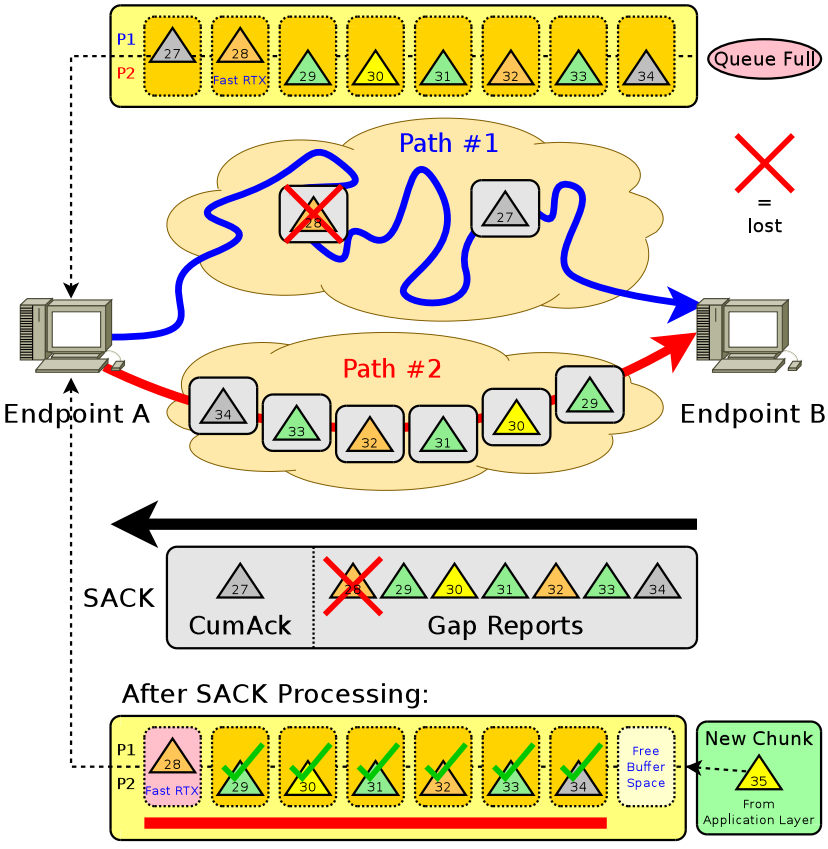


Figure 7.5: An Example for GapAck-Induced Send Buffer Blocking

the completely occupied send buffer, only *one* new DATA chunk may be enqueued and transmitted. That is, the sender cannot utilise its congestion window on Path #1; it is therefore unable to utilise the full capacity of Path #1.

This problem of unbalanced distribution of buffer space among the paths caused by transmission-induced send buffer blocking is particularly triggered by overly long router queues (here: on the high-delay Path #1), as will be demonstrated in an ADSL setup in Subsection 7.6.2. It is important to note here that it already occurs for unordered delivery, and therefore also for ordered delivery as well.

7.5.1.2 GapAck-Induced Send Buffer Blocking

The second sender-side buffer blocking issue is denoted as *GapAck-Induced Send Buffer Blocking*. Figure 7.5 presents an example: Endpoint A has transmitted the DATA chunks #27 and #28 on Path #1 as well as #29 to #34 on Path #2 using unordered delivery. On Path #1, the DATA chunk #27 is successfully received – as well as the DATA chunks #29 to #34 on Path #2. This allows a CumAck of TSN #27 and GapAcks for TSNs #29 to #34 (in form of a SACK chunk). The DATA chunk #28 is missing. Due to possible reneging, as described in Section 3.7, the succeeding DATA chunks #29 to #34 cannot be removed from the send buffer. Therefore, only the space of a single DATA chunk may be freed (by the CumAck of TSN #27), allowing only a single more new one to be enqueued and transmitted into the network.

In the worst case, DATA chunk #28 is a lost fast retransmission (i.e. it has already been transmitted twice). Then, once the buffer is fully blocked, the transmissions on *all* paths are suspended until a successful timer-based retransmission of this DATA chunk.

The NR-SACK extension – as described in Subsection 3.11.5 – is able to reduce the problem in certain situations. In the example, TSNs #29 to #34 may be non-renegably GapAck'ed, allowing the sender to remove the corresponding DATA chunks from its send buffer. However, while NR-SACK works for unordered delivery without Segmentation, a receiver may need reneging in certain other cases. Particularly, reneging may be necessary when using ordered delivery – as described in Section 3.7.

7.5.2 Receive Buffer Blocking

Receiver-side buffer blocking is denoted as *Receive Buffer Blocking*. It can also be classified into two sub-problems.

7.5.2.1 Advertised-Window-Induced Receive Buffer Blocking

An example for the first receiver-side buffer blocking issue – denoted as *Advertised-Window-Induced Receive Buffer Blocking* – is depicted in Figure 7.6: Endpoint B has told Endpoint A its advertised receiver window of eight DATA chunks. Therefore, Endpoint A has sent its eight DATA chunks #36 to #43 using unordered delivery to Endpoint B over two paths. After that, it has reduced the peer receiver window to 0 (as defined in Subsection 2.10.1), i.e. it currently may not send any further chunks.

Path #1 has a high delay and large queuing capacity. On the other hand, Path #2 is a high-bandwidth path with a low delay and a small queuing capacity. The DATA chunks #36 and #38 are the first chunks reaching Endpoint B; they may be delivered directly to the Application Layer. Using delayed acknowledgement (see Subsubsection 2.9.3.1), Endpoint B will then send a SACK chunk to Endpoint A, again with an advertised receiver window of eight DATA chunks. On reception of this

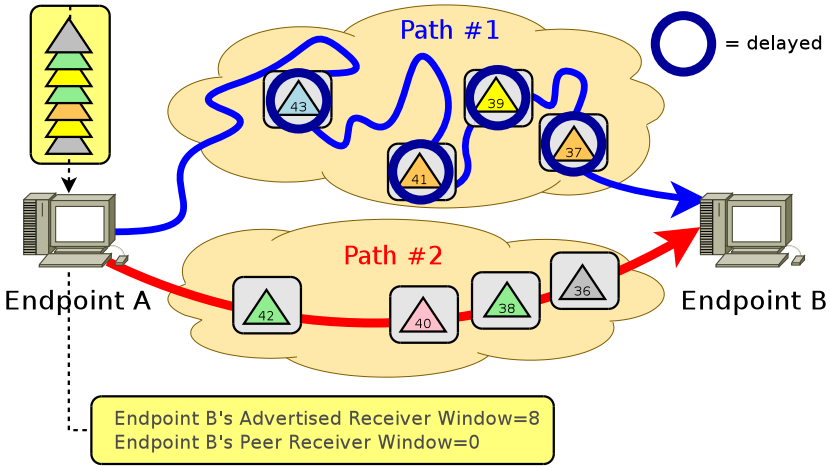


Figure 7.6: An Example for Advertised-Window-Induced Receive Buffer Blocking

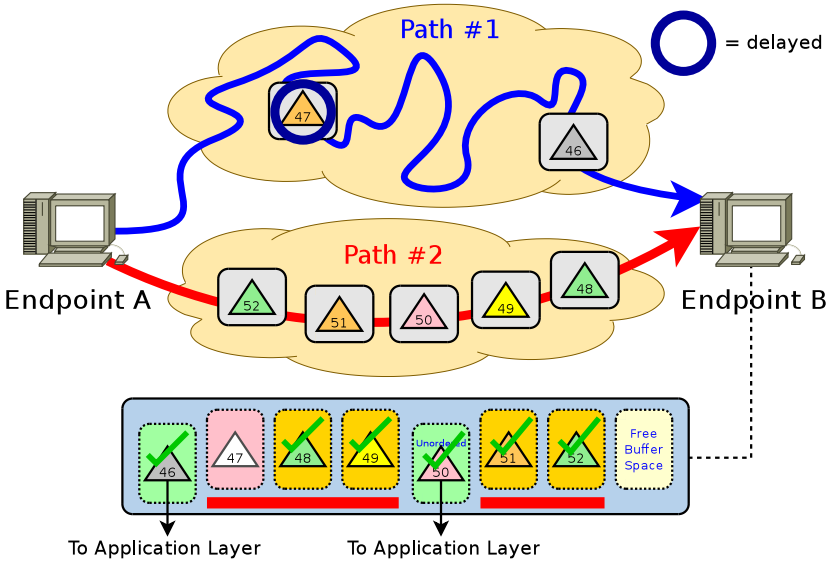


Figure 7.7: An Example for Reordering-Induced Receive Buffer Blocking

SACK chunk, Endpoint A knows that the TSNs #36 and #38 are not outstanding any more and adapts its peer receiver window to two DATA chunks (since six more chunks are still outstanding). According to the congestion window update rules, Endpoint A may also increase its congestion window for Path #2 (see Subsection 4.2.3). But the peer receiver window only allows Endpoint A to send two new DATA chunks into the network. That is, it will not be able to utilise the increased congestion window.

In order to fully utilise its capacity, a path needs to steadily have at least a number of bytes given by its bandwidth-RTT product outstanding (as described in Subsubsection 2.9.2.3). If it cannot have this amount of data in flight, its throughput will suffer. In the example depicted in Figure 7.6, this problem occurs since the queuing capacity of Path #1 is too large in comparison to the receive buffer capacity of Endpoint B – which is provided to Endpoint A as advertised receiver window. That is, Endpoint A has no possibility to send out further DATA chunks on Path #2 to utilise its capacity.

It is important to note here that advertised-window-induced receive buffer blocking may already occur for unordered delivery, as shown in the example. Therefore, it is a problem for ordered delivery as well. [IAS05] proposes specific retransmission strategies that can be used to alleviate some of the throughput reduction related to advertised-window-induced receive buffer blocking. However, they cannot solve the cause of the problem.

7.5.2.2 Reordering-Induced Receive Buffer Blocking

The second receiver-side buffer blocking issue is denoted as *Reordering-Induced Receive Buffer Blocking*; Figure 7.7 presents an example: similar to GapAck-induced send buffer blocking as described in Subsubsection 7.5.1.2, missing chunks preventing a CumAck lead to the issue of already-received DATA chunks blocking receive buffer space. In this example, Endpoint A has transmitted the DATA chunks #46 and #47 on Path #1 as well as #48 to #52 on Path #2. DATA chunk #47 is delayed; all other ones have arrived at Endpoint B.

Except for DATA chunk #50, all other chunks have used ordered delivery. That is, only the DATA chunk #50 – which uses unordered delivery – may be delivered out of sequence. Therefore, as soon as it has been received, it may be forwarded to the Application Layer and removed from the receive buffer. On the other hand, the DATA chunks #48 to #49 and #51 to #52 have to be provided to the Application Layer in sequence. For this reason, they must remain in the receive buffer until the still missing DATA chunk #47 has arrived. Until then, the corresponding buffer space of these four chunks remains occupied.

7.6 Buffer Splitting

The key problem of the observed buffer blocking problems has been the coupling – and therefore the resulting unbalance – of the send and receive buffer fractions used by the paths of a CMT-SCTP association.

7.6.1 The Approach

The approach, denoted as *Buffer Splitting*, which has been developed as part of this thesis, targets this unbalance by a decoupling of the per-path buffer usage.

7.6.1.1 Buffer Splitting based on Buffered Bytes

In the initial version of buffer splitting, which has been published in [DBRT10], the metric for the per-path usage of buffer space is the buffer size occupied by the DATA chunks on each path. In order to avoid one path using too much send buffer space – which prevents other paths from sending out new chunks – the solution denoted as *Send Buffer Splitting based on Buffered Bytes* simply splits the send buffer of size B^{Sender} into n (i.e. the number of paths) sections. Let Buffered_i be the buffer size occupied by chunks on path P_i and MTU_i be the MTU on path P_i. Then, a new DATA chunk on path P_i may be sent if its buffer share allows another MTU-sized packet:

$$\text{Buffered}_i + \text{MTU}_i \leq \frac{B^{\text{Sender}}}{n}. \tag{7.1}$$

Similar to the send buffer handling, the *sender* is also able to take care of the receive buffer, by taking notice of the peer receiver window size $\text{PeerReceiverWindow}$ (described in Subsection 2.10.1). Let Outstanding_j be the buffer size occupied by outstanding (i.e. still unacknowledged) chunks on path P_j. Using *Receive Buffer Splitting based on Buffered Bytes*, a new DATA chunk on path P_i may be sent if:

$$\text{Buffered}_i \leq \frac{\text{PeerReceiverWindow} + \sum_{j=1}^{n} \text{Outstanding}_j}{n}. \tag{7.2}$$

The peer receiver window describes the sender's current knowledge of available receive buffer space. Therefore, the total receive buffer size is simply the peer receiver window plus the size of the currently outstanding DATA chunks.

Note, that the approach of receive buffer splitting is completely realised at the sender side. That is, for *unidirectional* CMT-SCTP payload data transfer, the receiver side requires no particular support functionality. This may be useful in some deployment scenarios, where the SCTP implementation of the receiver side cannot be updated easily.

Since both, sender-side and receiver-side buffer splitting approaches are based on buffer occupation, they are commonly denoted as *Buffer Splitting based on Buffered Bytes*.

7.6.1.2 Buffer Splitting based on Outstanding Bytes

The approach of buffer splitting based on buffered bytes has been refined by reconsidering the unbalance of the buffer usage. Instead of using the complete buffer occupation of a path (i.e. all DATA chunks, regardless of whether they are outstanding or have already been acknowledged), it is already sufficient to just consider the number of outstanding bytes (see Subsection 2.9.2) on each path. This approach has been published in [ADB+11].

The approach of *Send Buffer Splitting based on Outstanding Bytes* simply modifies the precondition of Equation 7.1 to transmit a new DATA chunk on path P_i as follows:

$$\text{Outstanding}_i + \text{MTU}_i \leq \frac{B^{\text{Sender}}}{n} \tag{7.3}$$

Analogously, *Receive Buffer Splitting based on Outstanding Bytes* modifies the precondition of Equation 7.2 for the receive buffer handling as follows:

$$\text{Outstanding}_i \leq \frac{\text{PeerReceiverWindow} + \sum_{j=1}^{n} \text{Outstanding}_j}{n} \tag{7.4}$$

Both approaches – i.e. send and receive buffer splitting based on outstanding bytes – are further denoted as *Buffer Splitting based on Outstanding Bytes*. The difference to the approach based on buffer occupation will be further examined in Subsection 7.6.3.

7.6.2 A Proof of Concept

As a proof of concept for buffer splitting based on outstanding bytes, the setup described in Section 7.2 has been used with FIFO router queues, since RED is non-deterministic and would therefore blur the effects to be demonstrated. The maximum FIFO queue packet capacity κ (see Subsection 2.4.2) has been varied; Path #1 has used a fixed bandwidth of 100 Mbit/s, Path #2 a fixed bandwidth of 10 Mbit/s. Figure 7.8 presents the application payload throughput results for buffer splitting turned off (curve 3) as well as send buffer splitting (curve 1; see Equation 7.3), receive buffer splitting (curve 2; see Equation 7.4) and both simultaneously (curve 4), for three send buffer to receive buffer size ratios:

1. Using a send buffer smaller than the receive buffer (125,000 bytes vs. 250,000 bytes; shown in Subfigure 7.8(a)), the application payload throughput falls from the expected about 106 Mbit/s to just about 16 Mbit/s if no send buffer splitting is applied. Here, the queuing capacity of the slow Path #2 introduces transmission-induced send buffer blocking (as explained in Subsubsection 7.5.1.1) when it becomes too large. At κ=150 packets using an MTU of 1,500 bytes, this capacity has already reached 225,000 bytes – which is almost the size of the complete send buffer (that has to be shared by both paths). Send buffer splitting – alone (curve 1, drawn in brown colour) or in combination with receive buffer splitting (curve 4, drawn in orange colour) – solves the problem.

2. If the send buffer is larger than the receive buffer (250,000 bytes vs. 125,000 bytes; shown in Subfigure 7.8(b)), the scenario turns around. Now, advertised-window-induced receive buffer blocking (as explained in Subsubsection 7.5.2.1) occurs. Since the receive buffer – for both paths – only has a size of 125,000 bytes, the capacity of the queue on the slow Path #2 already exceeds it at κ=85 packets (i.e. 127,500 bytes). The growing number of outstanding DATA chunks on Path #2 limits the peer receiver window, leaving no more room to have enough outstanding DATA chunks on the fast Path #1 to cover the bandwidth-RTT product. Receive buffer splitting – alone (curve 2, drawn in magenta colour) or in combination with send buffer splitting (curve 4, drawn in orange colour) solves the problem here.

3. For equal buffer sizes (250,000 bytes for both; shown in Subfigure 7.8(c)), the problem is solved by send buffer splitting (curve 1, drawn in brown colour) *or* receive buffer splitting (curve 2, drawn in magenta colour): avoiding an unbalance in the send buffer also prevents an unbalance in the same-sized receive buffer, and vice versa. Clearly, the same effect is reached when applying both mechanisms simultaneously (curve 4, drawn in orange colour).

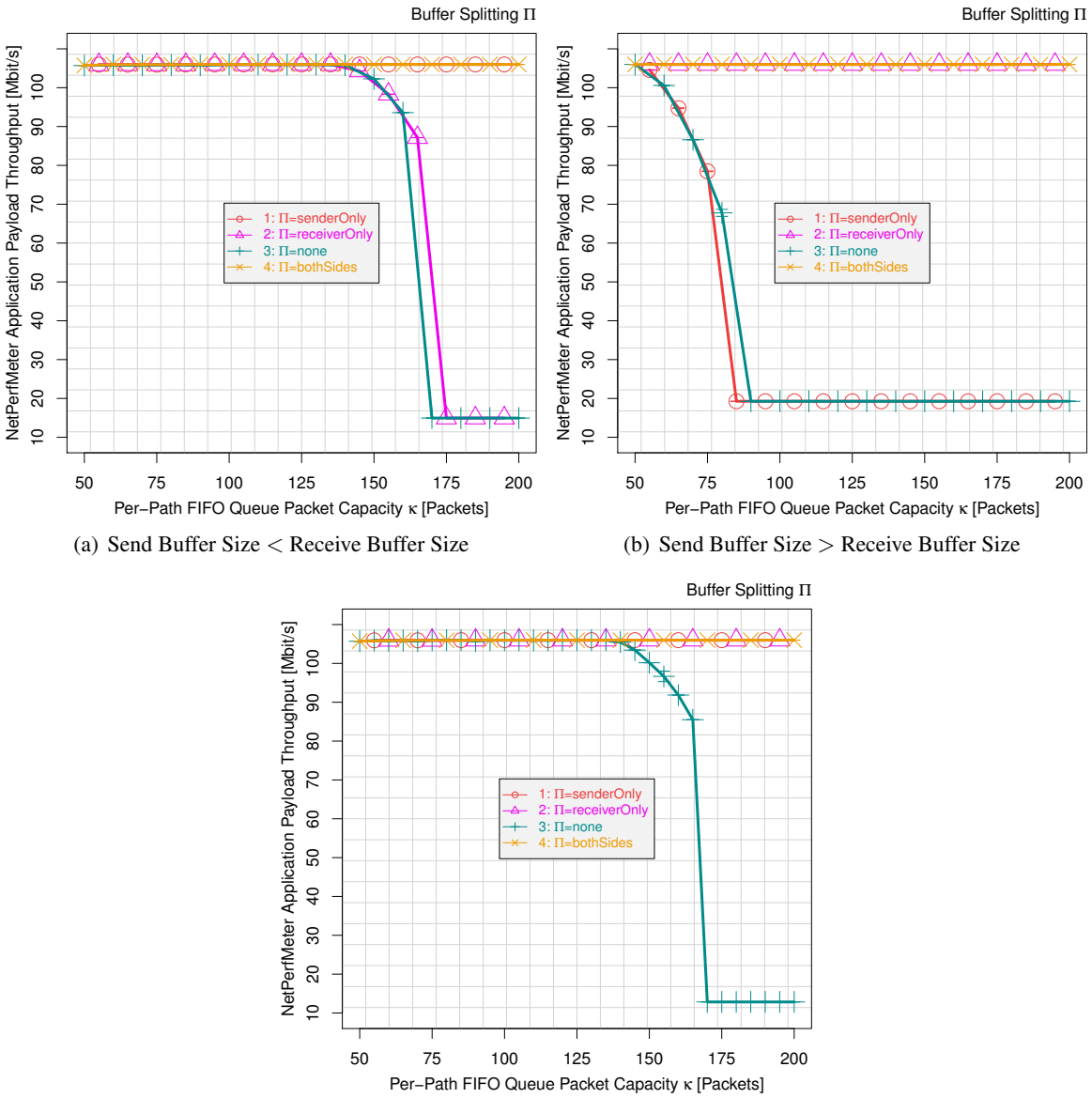
In summary, applying send *and* receive buffer splitting simultaneously (i.e. curve 4) solves the transmission-induced send buffer blocking and advertised-window-induced receive buffer blocking issues in all three cases.

7.6.3 Buffer Bloat – A Challenging Real-World Internet Scenario

In order to further examine the usefulness of buffer splitting, to show the difference between buffer splitting based on buffered bytes and buffer splitting based on outstanding bytes, and to also demonstrate the handling of GapAck-induced send buffer blocking, the simulation scenario for the following performance analysis is based on an ADSL setup.

7.6.3.1 The ADSL Scenario

The following parameters have been used:



(a) Send Buffer Size < Receive Buffer Size

(b) Send Buffer Size > Receive Buffer Size

(c) Send Buffer Size = Receive Buffer Size

Figure 7.8: A Proof of Concept for Buffer Splitting based on Outstanding Bytes

- The message size has been 1,444 bytes at an MTU of 1,492 bytes on the ADSL links (i.e. MTU-sized packets). The reduced MTU size – instead of 1,500 bytes as for Ethernet – is caused by the usage of PPPoE for the ADSL communication (as described in Subsection 2.13.1).
- The configurations of Path #1 and Path #2 have been based on the distributed testbed setup described in Subsection 6.5.2:
  - Path #1 has been a high-speed path with a configurable bandwidth ρ_{HS} and a delay of 2 ms (i.e. the RTT has been 4 ms).
  - Path #2 has been an ADSL path. Its bandwidth has been limited to ρ_{ADSL}=800 Kbit/s, its delay has been 28 ms (i.e. the RTT has been 56 ms). Furthermore, this path has used a FIFO queue with a capacity of 100 packets (the reason will be described below).
- The send buffer has been set to 300,000 bytes, the receive buffer has been set to 100,000 bytes.

The reason for using an ADSL path is the challenge introduced by so-called *Buffer Bloat*, as described by [Get11a, Get11b]. ADSL modems typically use very long FIFO queues. These long queues intend to maximise transfer throughputs (which is a nice feature for marketing purposes in the highly competitive ADSL ISP market), but also have a negative impact on the delay. Having a queue size of 100 packets at a speed of 800 Kbit/s, the queuing capacity for packets of 1,492 bytes is:

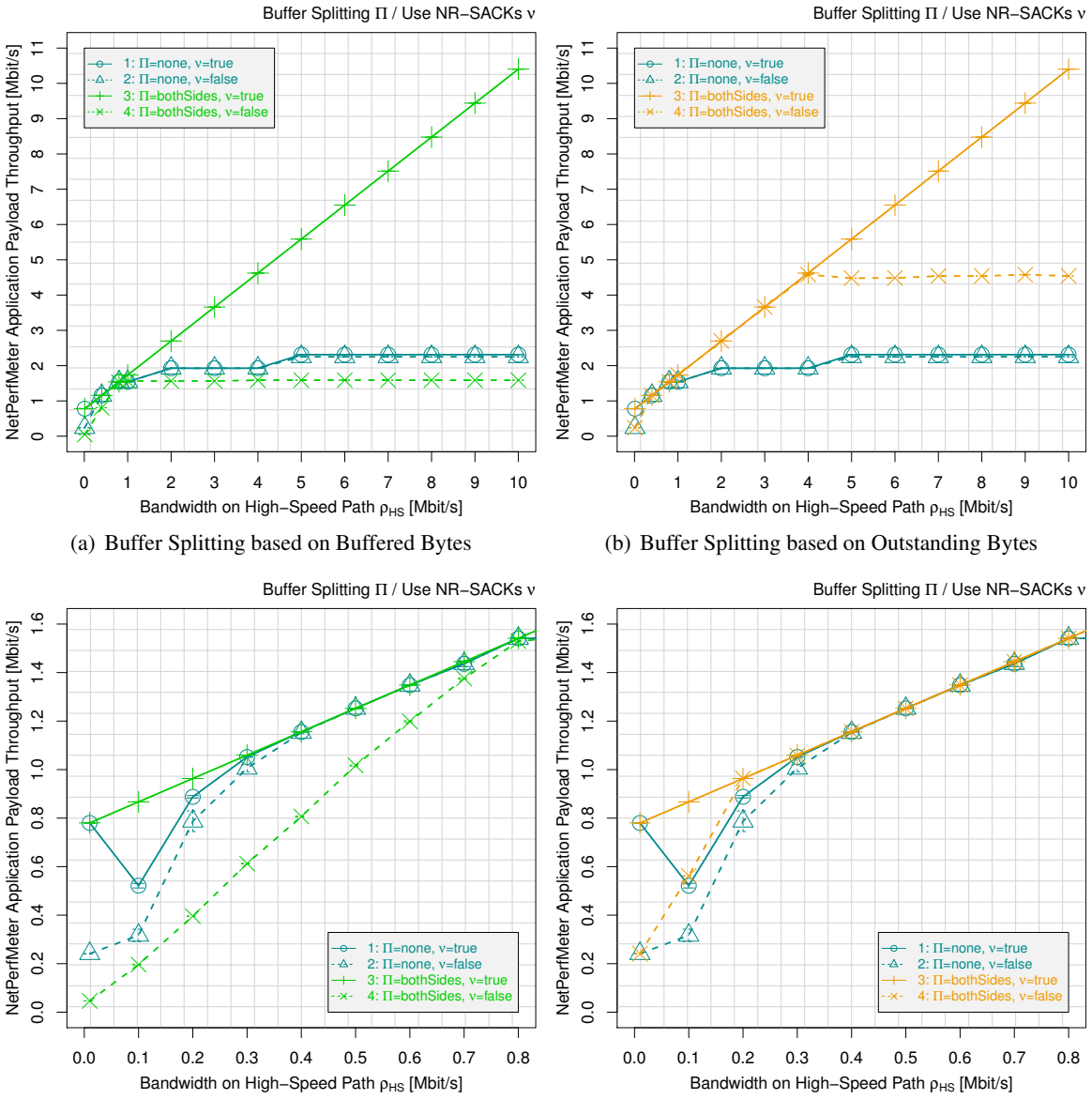
$$\frac{100 \text{ packets} * 1,492 \text{ B/packet}}{\frac{800{,}000 \text{ bit/s}}{8 \text{ bit/B}}} \approx 1.5 \text{ s}.$$

This means that a fully-utilised queue adds an additional delay of 1.5 s – which is e.g. more than six times the delay caused by a satellite link (distance of 2×36,000 km; see also Subsection 2.4.2). The impact on a window-based congestion control, as described in Subsection 2.11.2, is an extreme growth of the congestion window. Once the congestion window has reached the bandwidth-RTT product (see Subsubsection 2.9.2.3), the throughput of the communication cannot be improved any more. Any larger setting of the congestion window just increases the message delay.

7.6.3.2 Simulation Results

In order to evaluate the impact of the ADSL setup on the CMT-SCTP performance, the bandwidth ρ_{HS} of the high-speed path (i.e. Path #1) has been varied from 10 Kbit/s to 10 Mbit/s. The resulting application payload throughput results are presented in Figure 7.9. Subfigure 7.9(a) shows the results for applying buffer splitting based on buffered bytes (as defined in Subsubsection 7.6.1.1), Subfigure 7.9(b) presents the corresponding results for applying buffer splitting based on outstanding bytes (as defined in Subsubsection 7.6.1.2). Since small settings of ρ_{HS} – resulting in a high dissimilarity of the paths – are also interesting, the plots in Subfigure 7.9(c) and Subfigure 7.9(d) show an extract of the results for ρ_{HS}=10 Kbit/s to ρ_{HS}=800 Kbit/s. The curves 1 and 2 (drawn in cyan colour) present the results for buffer splitting turned off; the curves 3 and 4 show the performance for activated buffer splitting. The impact of deactivating the NR-SACK extension is depicted by curves 2 and 4 (drawn as dashed lines); otherwise, NR-SACK has been used (drawn as solid lines).

Obviously, if buffer splitting is turned off, the sender is unable to utilise the high-speed path by having a sufficient number of outstanding DATA chunks in order to cover the bandwidth-RTT product. Advertised-window-induced receive buffer blocking – as described in Subsubsection 7.5.2.1 – is caused by the long transmission queue on the ADSL path. The resulting buffer bloat leads to



(a) Buffer Splitting based on Buffered Bytes

(b) Buffer Splitting based on Outstanding Bytes

(c) Buffer Splitting based on Buffered Bytes (Extracted)

(d) Buffer Splitting based on Outstanding Bytes (Extracted)

Figure 7.9: Simulation Results for the Impact of Buffer Splitting in the ADSL Scenario

a large number of outstanding bytes on the slow ADSL path, leaving no more room for increasing the outstanding bytes on the high-speed path. The outstanding bytes are limited by the advertised receiver window (the receive buffer has a size of 100,000 bytes), which is clearly much smaller than the size of the send buffer (300,000 bytes). This size difference is also the reason why NR-SACK cannot improve the situation here: NR-SACK can help to reduce the send buffer space requirements (see Subsection 3.11.5), but the send buffer is not the problem here.

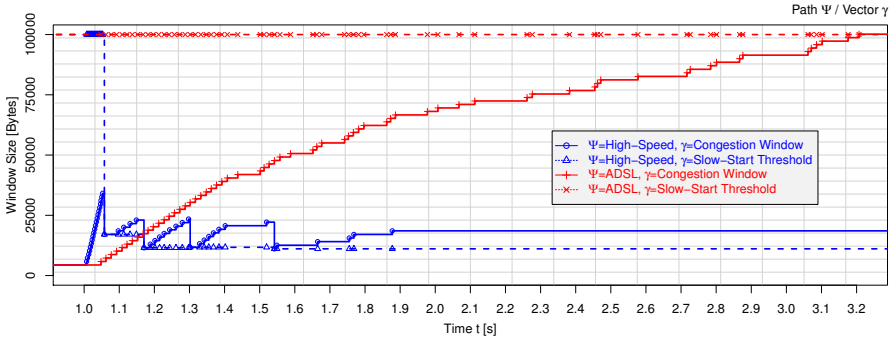
The usage of buffer splitting – in either variant – leads to a significantly increased performance. If combined with the usage of the NR-SACK extension (i.e. curve 3), both buffer splitting variants reach the expected application payload throughput of nearly 10.4 Mbit/s at ρ_{HS}=10 Mbit/s. If NR-SACK is turned off (i.e. curve 4), the throughput is significantly lower in more dissimilar path scenarios (i.e. for ρ_{HS}<800 Kbit/s and ρ_{HS}>800 Kbit/s). Obviously, GapAck- and transmission-induced send buffer blocking (see Subsubsection 7.5.1.2 and Subsubsection 7.5.1.1) occurs and prevents the high-speed path from utilising its bandwidth. As introduced in Subsubsection 7.6.1.2, this problem is clearly stronger when basing the buffer splitting on buffered instead of outstanding bytes. That is, the improved variant of buffer splitting (based on outstanding bytes) achieves a significant performance improvement when NR-SACK cannot be applied, e.g. if the receiver side does not support this protocol extension. Unlike buffer splitting, which is completely realised at the sender side, NR-SACK requires support by sender and receiver (see also Subsection 3.11.5). However, the GapAck-induced send buffer blocking can only be solved by applying the NR-SACK extension.

7.6.3.3 Impact on the Congestion Control Behaviour

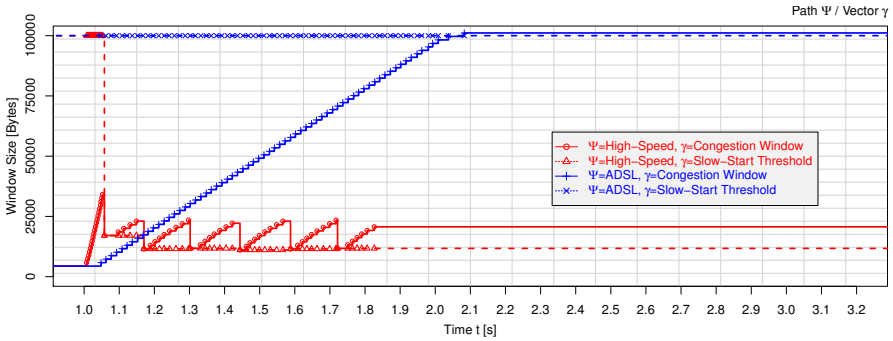
In order to further explain the effects causing the throughput results shown above, Figure 7.10 presents an extract from time t_0=1 s to t_1=3.2 s of the corresponding congestion window (drawn as solid lines) and slow-start threshold (drawn as dashed lines) behaviour of the high-speed path (Path #1, shown in blue colour) and the ADSL path (Path #2, shown in red colour) for each of the four cases (i.e. send and receive buffer splitting based on outstanding bytes off/on; NR-SACKs off/on). The bandwidth of the high-speed path ρ_{HS} has been set to 10 Mbit/s here.

Subfigure 7.10(a) shows the results for buffer splitting as well as NR-SACKs turned off. Obviously, the congestion window of the ADSL path steadily increases to almost the size of the advertised receiver window (i.e. 100,000 bytes). As long as there is room in the receive window, the saturated sender tries to increase the congestion window of a path on reception of a new acknowledgement, as described in Subsubsection 2.11.2.1. The long ADSL queue of up to 100 packets leads to a linearly increased message delay on growing queue occupation. Here, the rising number of outstanding bytes on the ADSL path fills this queue – which causes a higher delay – but no further improvement of the application payload throughput. Even worse, since the high-speed path is bandwidth-limited, the congestion window of the ADSL path may take more buffer space when the high-speed path experiences a packet loss, which occurs regularly as part of the normal AIMD behaviour (as introduced in Subsection 2.11.3). In the end, the ADSL path – experiencing no loss due to its long queue – almost fully occupies the send buffer. This implies no possibility for the congestion window of the high-speed path to grow large enough to cover at least the bandwidth-RTT product in order to fully utilise the capacity of its path.

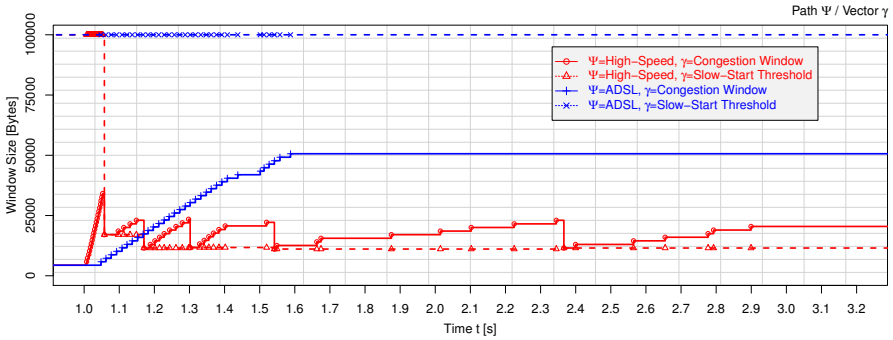
Only applying the NR-SACK extension has the effect that the congestion window growth for the ADSL path becomes almost perfectly linear, as shown in Subfigure 7.10(b). Without the need to wait for a CumAck, the sender may increase the outstanding bytes immediately on reception of an acknowledgement, i.e. GapAck-induced send buffer blocking (as described in Subsubsection 7.5.1.2) is avoided. Once the congestion window of the ADSL path is large enough – here for $t \geq$1.8 s – it



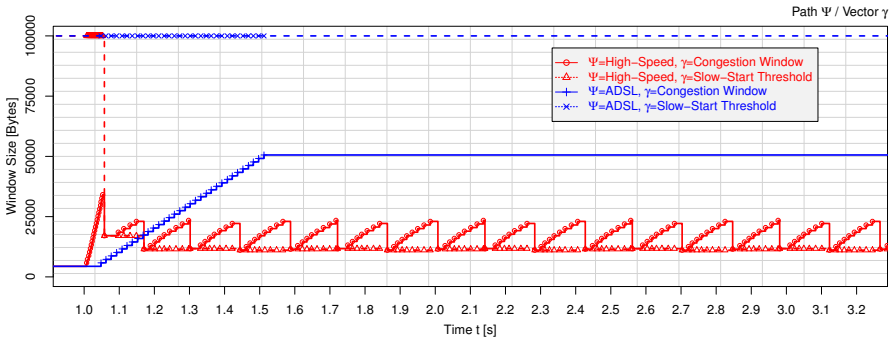
(a) Without Buffer Splitting, Without NR-SACK



(b) Without Buffer Splitting, With NR-SACK



(c) Both-Side Buffer Splitting, Without NR-SACK



(d) Both-Side Buffer Splitting, With NR-SACK

Figure 7.10: The Impact of Buffer Splitting and NR-SACKs on the Congestion Control Behaviour

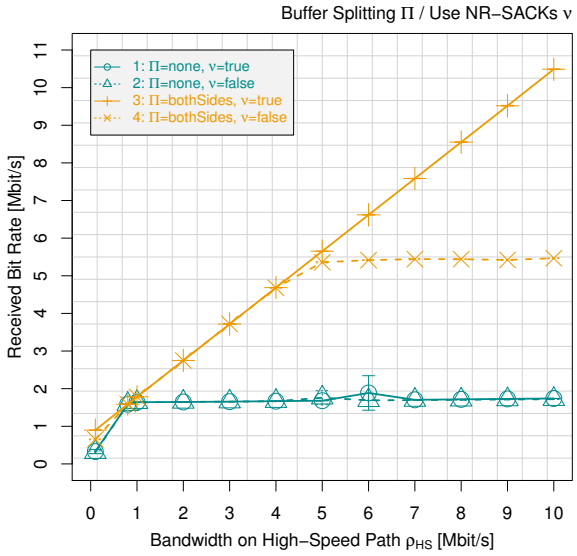


Figure 7.11: Experimental Validation of Buffer Splitting in the Distributed Testbed Setup

causes advertised-window-induced receive buffer blocking (as described in Subsubsection 7.5.2.1), which makes the typical AIMD behaviour on the high-speed path (i.e. growing until a loss, then restarting from the slow-start threshold) impossible.

Subfigure 7.10(c) presents the results for only using buffer splitting based on outstanding bytes, but without NR-SACKs. Clearly, the congestion window of the ADSL path can now only take a send buffer space of at most half of the receive window (i.e. 50,000 bytes) – which would in fact leave enough room for increasing the number of outstanding bytes on the high-speed path. However, while the throughput of this path is actually increased (as shown by curve 4 in Subfigure 7.9(b)), the congestion window curve of the high-speed path does not show the typical AIMD behaviour. The reason is GapAck-induced send buffer blocking due to the need to wait for TSNs on the long-delay[3] ADSL path to be acknowledged, in order to CumAck a sequence of chunks. This CumAck is mandatory in order to actually gain space in the send buffer for sending new DATA chunks.

Finally, when combining buffer splitting with NR-SACKs, the GapAck-induced send buffer blocking is solved, as shown in Subfigure 7.10(d). The feature of non-renegably acknowledging chunks allows the sender to remove selectively – but not yet cumulatively – acknowledged chunks from the send buffer and to gain the space required to put further DATA chunks in flight. This leads to fully utilising both paths and achieving the expected payload throughput (see curve 3 in Figure 7.9).

7.6.3.4 From Simulation to Reality

Clearly, it is useful to also validate the results obtained by simulations in reality. Therefore, a measurement corresponding to the simulation scenario has been performed in the distributed testbed setup described in Subsection 6.5.2. Figure 7.11 shows the achieved application payload throughput in the four configuration cases: buffer splitting based on outstanding bytes on both sides turned off (i.e. curves 1 and 2 drawn in cyan colour) or on (i.e. curves 3 and 4 drawn in orange colour) as well as NR-SACKs turned on (i.e. curves 1 and 3 drawn as solid lines) or off (i.e. curves 2 and 4 drawn as

[3]This delay is particularly high, due to the filling of the ADSL queue.

dashed lines).

Comparing the measurement results to the simulation results presented in Subfigure 7.9(b), the reality matches the expectations from the simulation results quite well. The payload throughput linearly increases with the growing high-speed path bandwidth ρ_{HS} if combining buffer splitting with the usage of the NR-SACK extension (curve 3). Just applying buffer splitting alone (i.e. without NR-SACKs; curve 4) still achieves a similar performance improvement, up to a certain dissimilarity of the paths (here: $\rho_{\mathrm{HS}} \geq 4$ Mbit/s). This improvement is slightly higher than for the simulation, due to Internet background congestion on the ADSL path. Since the real ADSL path sometimes experiences losses which are not caused by the ADSL modem queue, it has to reduce its congestion window more frequently, leaving a little more room for the other path. Without buffer splitting, only a quite constant payload throughput of less than 2 Mbit/s is achieved – regardless of turning NR-SACKs on (curve 1) or off (curve 2).

In summary, the proof of concept and the ADSL testbed scenario have shown that buffer splitting is useful to avoid the issues of

- Transmission-induced send buffer blocking and
- Advertised-window-induced receive buffer blocking.

Furthermore, the NR-SACK extension prevents GapAck-induced send buffer blocking.

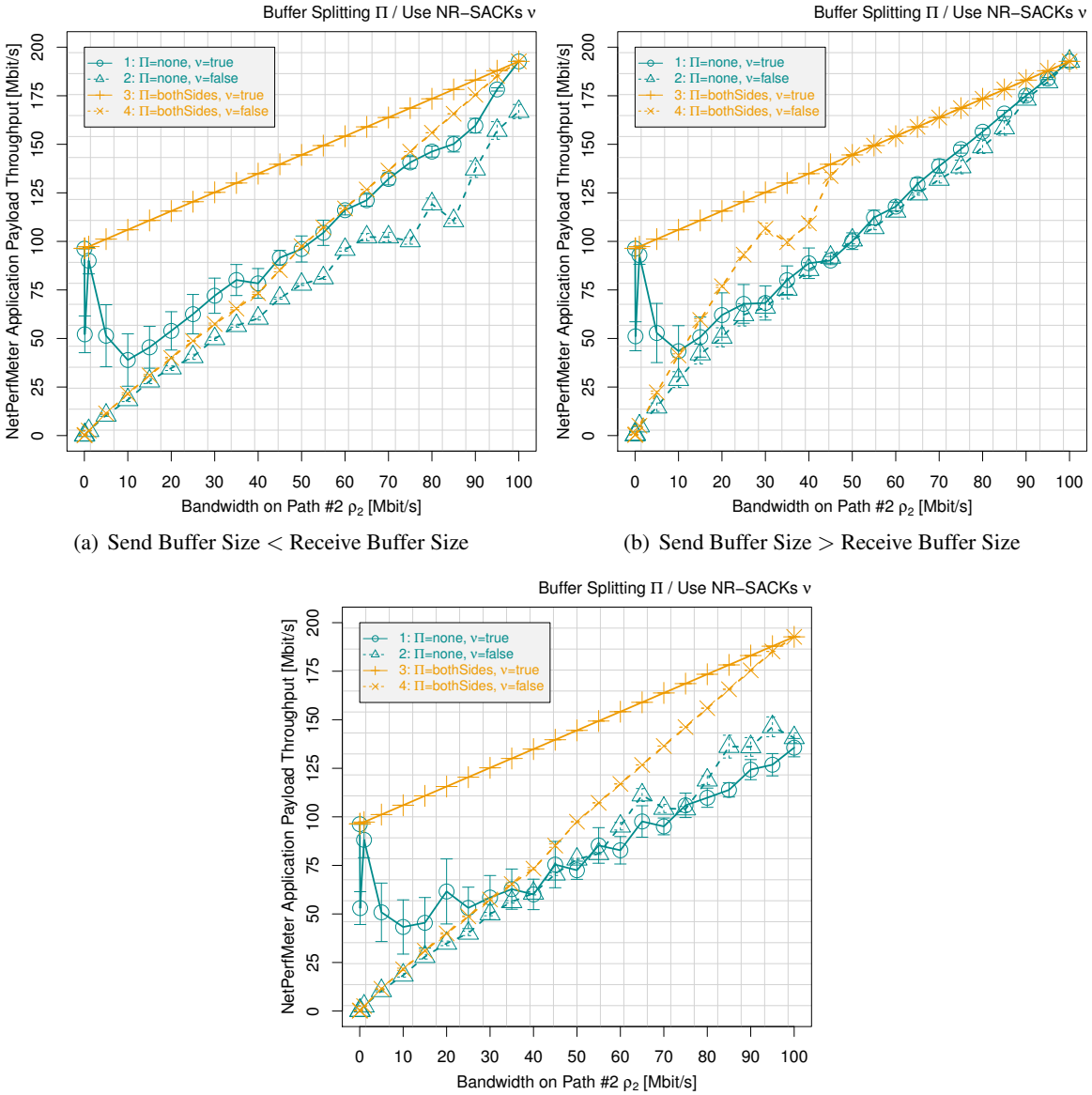
7.7 Unordered Delivery

In order to show how to provide efficient unordered delivery, a simulative parameter study has been performed, covering a large parameter space of the basic QoS characteristics (as described in Section 2.4) for path dissimilarity, i.e. bandwidth, bit error rate and delay. Note, that varying one of these parameters also has an influence on the other parameters: changing the bandwidth of a path also adapts its delay (due to queuing and buffer bloat; see Subsection 7.6.3), a higher bit error rate (i.e. packet losses, which are assumed as sign of congestion; see Subsection 2.11.2) reduces the achievable throughput (i.e. usable bandwidth), etc.. The initial version of the following parameter study has also been published in [DBRT10].

7.7.1 Dissimilar Bandwidths

In order to show the impact of bandwidth dissimilarity, the bandwidth of Path #2 has been varied from ρ_2=10 Kbit/s (i.e. 0.01 Mbit/s) to ρ_2=100 Mbit/s, while keeping the bandwidth of Path #1 constant at ρ_1=100 Mbit/s. Figure 7.12 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios. Subfigure 7.12(d) shows the actual buffer sizes, Subfigure 7.12(e) presents the used combinations of mechanisms (i.e. both-side buffer splitting based on outstanding bytes and NR-SACK) and their corresponding point symbols and curve numbers. Note, that the throughput of 2×100 Mbit/s is $2.5 * 10^7$ bytes/s, which is more than 23.8 MiB/s. That is, the buffer sizes are by two orders of magnitude smaller, as motivated in detail in Section 7.4.

For using neither buffer splitting nor NR-SACKs (curve 2), the performance is as expected. GapAck-induced and transmission-induced send buffer blocking in combination with advertised-window-induced receive buffer blocking result in a rather low performance. At a dissimilarity as small as 100/50 Mbit/s, the application payload throughput performance hardly reaches the expected performance for a non-CMT setup on Path #1 (which is about 95 Mbit/s, see also Section 7.3) for send buffer size $<$ receive buffer size (see Subfigure 7.12(a)) and send buffer size $>$ receive buffer size (see



(a) Send Buffer Size < Receive Buffer Size

(b) Send Buffer Size > Receive Buffer Size

(c) Send Buffer Size = Receive Buffer Size

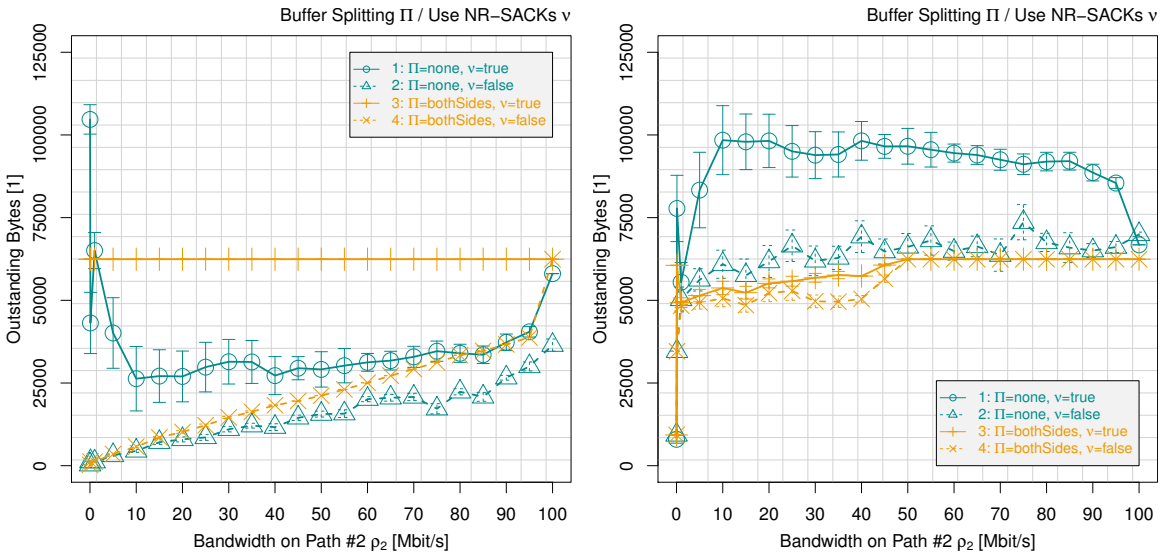
Figure	Send Buffer	Receive Buffer
7.12(a)	125,000 B	250,000 B
7.13		
7.12(c)	125,000 B	125,000 B
7.12(b)	250,000 B	125,000 B

(d) Buffer Size Configuration

Buffer Splitting	**NR-SACK**	
	disabled	enabled
disabled	△ (2)	○ (1)
enabled	× (4)	+ (3)

(e) Features Configuration

Figure 7.12: Throughput for Unordered Delivery over Paths with Dissimilar Bandwidths



(a) Path #1 (b) Path #2

Figure 7.13: Outstanding Bytes for Unordered Delivery over Paths with Dissimilar Bandwidths

Subfigure 7.12(b)). For send buffer size = receive buffer size (see Subfigure 7.12(c)), the situation is even worse. Here, also the similar path setup (i.e. 100/100 Mbit/s) only reaches a throughput of only about 130 Mbit/s.

Turning on the support for NR-SACKs (curve 1), the problem of GapAck-induced send buffer blocking is solved. This is particularly useful when the send buffer size is smaller than the receive buffer size (see Subfigure 7.12(a)), leading to the expected performance improvement. GapAck-induced send buffer blocking is particularly problematic for small settings of ρ_2, where the queue size on Path #2 (MinTh=30, i.e. for up to 30 packets, the used RED queue behaves like a FIFO queue; see also Subsection 2.4.2) leads to the issue of buffer bloat, as explained in Subsection 7.6.3. For example, the additional delay caused by a queue loaded with 30 packets for ρ_2=100 Kbit/s is:

$$\frac{30 \text{ packets} * 1,500 \text{ B/packet}}{\frac{100{,}000 \text{ bit/s}}{8 \text{ bit/B}}} = 3.6 \text{ s}.$$

Using buffer splitting, but without NR-SACKs – as presented by curve 4 – solves the problem of transmission-induced send buffer blocking in combination with advertised-window-induced receive buffer blocking. This gives a significant performance improvement over a scenario with both mechanisms turned off (i.e. in comparison to curve 2), but the higher the dissimilarity, the smaller the improvement. At about $\rho_2 \leq$20 Mbit/s, the effect of GapAck-induced send buffer blocking becomes the main issue, which cannot be solved by buffer splitting.

The problems of dissimilar bandwidths can only be solved by using buffer splitting and the NR-SACK extension simultaneously (curve 3), which results in a linear performance increase with the expected application payload throughput results – over the whole, large parameter range of ρ_2.

In order to further show the effects of buffer splitting and NR-SACKs, Figure 7.13 shows the outstanding bytes on Path #1 (Subfigure 7.13(a)) and Path #2 (Subfigure 7.13(b)) for the send buffer size of 125,000 bytes and the receive buffer size of 250,000 bytes. Since the other two scenarios

produce similar results, the corresponding plots have been omitted due to space reasons. While the outstanding bytes on the fast Path #1 (i.e. ρ_1=100 Mbit/s) without NR-SACKs drop to almost zero for small settings of ρ_2 (curves 2 and 4), they remain constant at 62,500 bytes (i.e. $\frac{125{,}000}{2}$) when NR-SACKs as well as buffer splitting are turned on (curve 3). In this case, the bandwidth-RTT product – as described in Subsubsection 2.9.2.3 – is covered and the fast Path #1 can utilise its full bandwidth.

Just using NR-SACKs alone only leads to a reasonable value for the outstanding bytes in cases of extreme dissimilarity, where the congestion window of Path #2 – due to the buffer bloat – cannot grow quickly enough to almost fully block Path #1. In this case, i.e. for ρ_2<10 Mbit/s, the outstanding bytes on Path #2 decrease (see Subfigure 7.13(b)), giving Path #1 the possibility to increase its outstanding bytes (see Subfigure 7.13(a)). In result, due to the now better utilisation of the high-bandwidth Path #1, the application payload throughput improves to almost the expected values (see Figure 7.12) in many cases. However, in some other cases, like for the negative peak at ρ_2=0.1 Mbit/s, an awkward combination of conditions leads to a performance drop by introducing an unbalance to the number of outstanding bytes (see Figure 7.13).

7.7.2 Dissimilar Bit Error Rates

For presenting the impact of bit error rate dissimilarity, the bit error rate of Path #2 has been varied from ϵ_2=0 to ϵ_2=$5 * 10^{-6}$ errors/bit, while keeping Path #1 error-free (i.e. ϵ_1=0). Figure 7.14 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios. Subfigure 7.14(d) shows the buffer size configurations and Subfigure 7.14(e) the used combination of mechanisms (both-side buffer splitting based on outstanding bytes, NR-SACK and the packet drop reporting extension introduced in Subsection 3.11.8) with their corresponding point symbols and curve numbers.

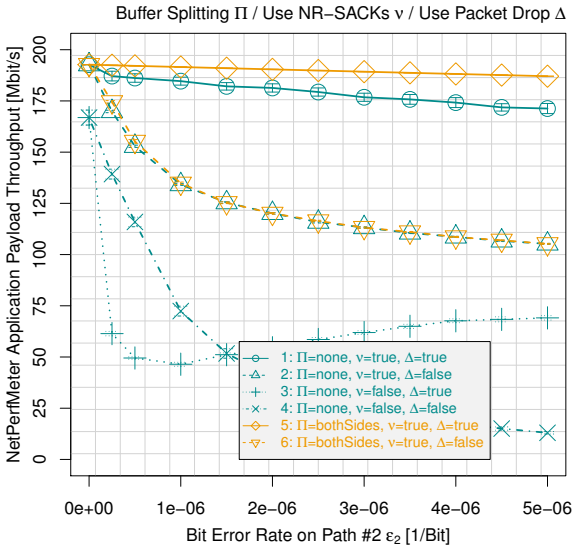
For example, at an MTU of 1,500 bytes, bit error rates of $1 * 10^{-6}$ and $5 * 10^{-6}$ errors/bit mean average full-size packet damage rates of:

$$\begin{aligned} 1 * 10^{-6}\ \text{errors/bit} * 8\ \text{bit/B} * 1{,}500\ \text{B/packet} &= 1.2\%, \\ 5 * 10^{-6}\ \text{errors/bit} * 8\ \text{bit/B} * 1{,}500\ \text{B/packet} &= 6.0\%. \end{aligned}$$

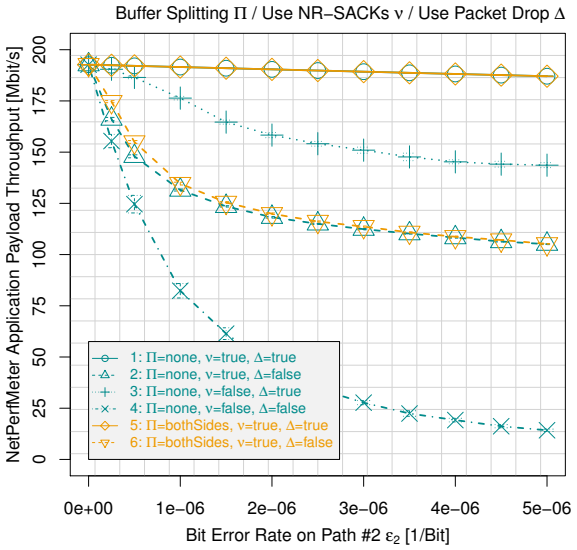
That is, about every 83rd or 16th packet, respectively, is damaged. Note, that these error rates – while being unlikely for properly-working cabled networks – may be realistic for wireless networks in presence of interferences or under poor reception conditions.

Clearly, a significant performance problem is caused by GapAck-induced send buffer blocking when NR-SACKs as well as buffer splitting are turned off (i.e. curves 3 and 4). At ϵ_2=$1 * 10^{-6}$, only the curve 3 – for packet drop reporting activated – in the scenario of send buffer size > receive buffer size (see Subfigure 7.14(b)) can exceed the non-CMT ideal-case (i.e. loss-free) application payload throughput performance of about 95 Mbit/s. This scenario – with its larger send buffer – is clearly less susceptible for the send buffer blocking issue. All other scenarios have a significantly lower throughput.

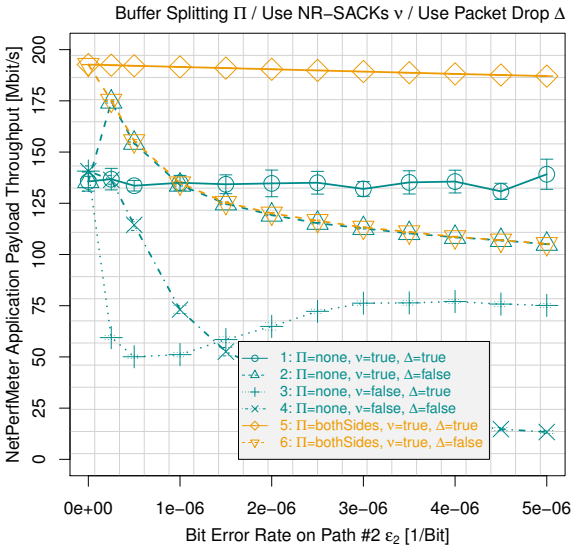
Note the interesting behaviour of curve 3 in Subfigure 7.14(a) and Subfigure 7.14(c). Here, the application payload throughput falls to a minimum at about $\epsilon_2 = 1 * 10^{-6}$ errors/bit and starts rising again for higher settings of ϵ_2. The reason for this effect is that the rising number of reported packet drops slightly countervails the effect of transmission-induced send buffer blocking (which is prevalent in the scenarios of send buffer size $\leq$ receive buffer size) by temporarily clearing space in the peer receiver window. Since transmissions are handled in a round-robin manner among the paths, this space may be used for a packet on the other path first (i.e. before the retransmission of the dropped chunk).



(a) Send Buffer Size < Receive Buffer Size



(b) Send Buffer Size > Receive Buffer Size



(c) Send Buffer Size = Receive Buffer Size

Figure	Send Buffer	Receive Buffer
7.14(a)	125,000 B	250,000 B
7.14(c)	125,000 B	125,000 B
7.14(b)	250,000 B	125,000 B

(d) Buffer Size Configuration

Buffer Splitting	**Packet Drop Rep.**	
	disabled	enabled
disabled, SACK only	× (4)	+ (3)
disabled, **NR-SACK**	△ (2)	○ (1)
enabled, **NR-SACK**	▽ (6)	◇ (5)

(e) Features Configuration

Figure 7.14: Throughput for Unordered Delivery over Paths with Dissimilar Bit Error Rates

Turning the support for NR-SACKs on, a significant performance gain is reached compared to the previous scenarios. For send buffer size > receive buffer size (see Subfigure 7.14(b)) and using packet drop reporting (i.e. curve 1; it is superimposed by curve 5 in this plot), the performance even reaches the expected value. However, for smaller settings of the send buffer size, transmission-induced send buffer blocking prevents the full utilisation of the bandwidth (see curve 1 in Subfigure 7.14(a) and Subfigure 7.14(c)). Also, note the – in comparison to the other curves – larger confidence intervals of curve 1, which indicate a significant variance of the achieved throughput values.

Obviously, the best performance is reached by turning buffer splitting as well as the NR-SACK extension on simultaneously. When not using packet drop reporting (i.e. curve 6), the bandwidth of the error-prone Path #2 cannot be fully utilised. Packet losses due to corruption leads to a reduction of the congestion window, as explained in Subsection 2.11.2. Nevertheless, even for a bit error rate as high as ϵ_2=5 $*$ 10^{-6} errors/bit, the achieved application payload throughput performance is still about 105 Mbit/s, which is by about 10 Mbit/s better than the non-CMT throughput on Path #1 alone. Clearly, in the ideal case, packet drop reporting is supported. Then, even such a high bit error rate only slightly decreases the total throughput to just about 185 Mbit/s. However, in order to achieve this performance, the packet drop reporting extension requires that corrupted packets are still delivered[4] to the destination endpoint, in order to report packets as corrupted. Alternatively, intermediate nodes (i.e. routers) may generate such reports, which – of course – requires support for SCTP packet drop reporting by these nodes as well.

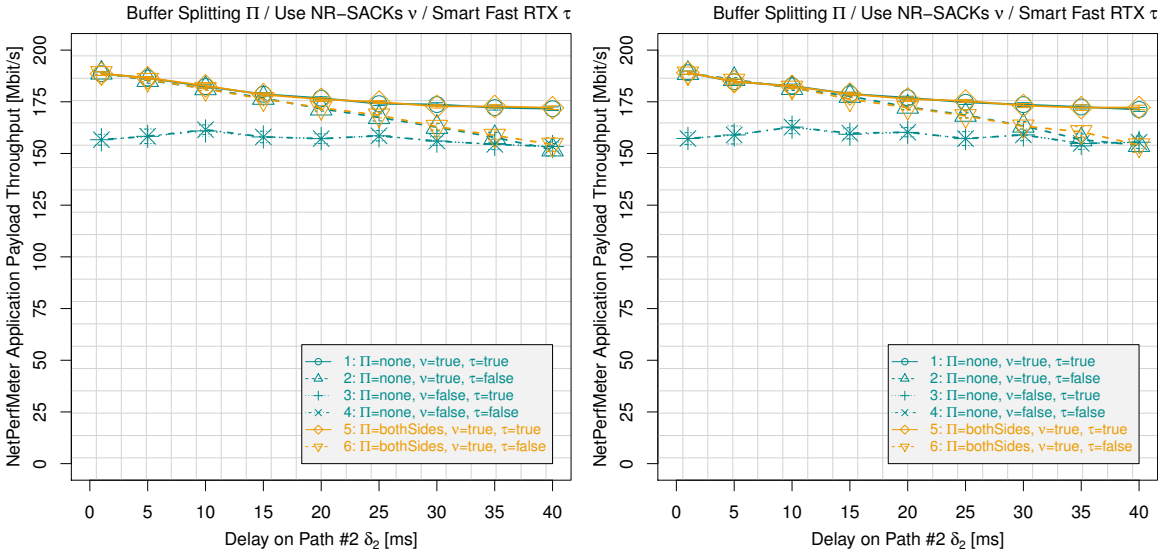
7.7.3 Dissimilar Delays

When paths have different delays, the occasional appearance of timer-based retransmissions – as explained in Subsubsection 2.9.3.4 – can cause a problem: according to RFC 4960 (see [Ste07, Subsubsection 6.3.3]), *any* outstanding DATA chunk on the affected path "should be marked for retransmission". Unlike fast retransmissions, timer-based retransmissions are rare and seen as sign of problems (like severe congestion) on a path. Therefore, if multiple paths are available, a timer-based retransmission is made on an alternative path. This behaviour is useful, since it shifts traffic away from a possibly broken path to a working one.

But if such an outstanding DATA chunk with TSN c has just been delayed and its timer-based retransmission has been made rashly, DATA chunk c exists in the network twice. That is, it exists on the original path and – as retransmission – on the alternative one. If the original transmission of c gets acknowledged, it appears to the sender as an acknowledgement of c on the *new* path (i.e. the alternative one), since the original transmission of TSN c has been assumed as lost. The problem arises if c somewhat differs from the TSN range on the new path (i.e. its TSN is lower or higher, due to the delay difference between the paths). Then, the CMT-SCTP fast retransmission handling – as described in Subsection 4.2.2 – may assume large gaps in the TSN sequence of the new path. These putative gaps trigger bursts of unnecessary fast retransmissions. While the resulting duplicate DATA chunks are simply dropped by the receiver, the fast retransmissions lead to a congestion window reduction – which in result reduces the throughput of the corresponding path. [IAS06] proposes some heuristics to cope with this problem.

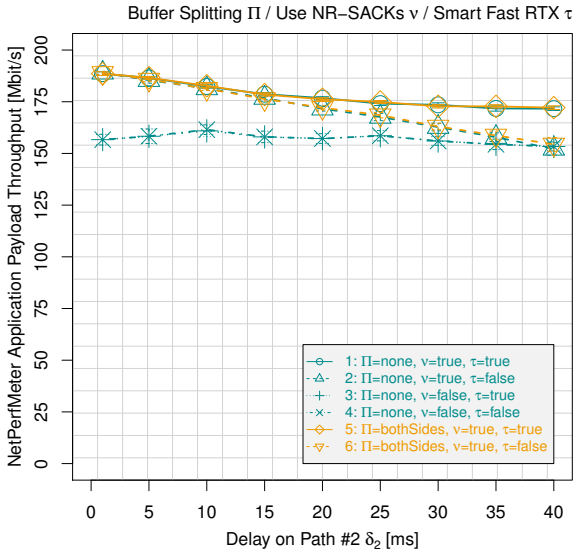
Instead of using heuristics, which are difficult to manage and implement, a simpler and easier to realise approach – denoted as *Smart Fast Retransmission* – has been developed as part of this thesis and published in [DBRT10]. It simply does not consider DATA chunks that have been moved from another path in the decision about fast retransmissions on their new path.

[4] Of course, the destination information inside IP and SCTP headers must still be intact. See also Subsubsection 2.9.3.5.



(a) Send Buffer Size < Receive Buffer Size

(b) Send Buffer Size > Receive Buffer Size



(c) Send Buffer Size = Receive Buffer Size

Figure	Send Buffer	Receive Buffer
7.15(a)	1,250,000 B	2,500,000 B
7.15(c)	1,250,000 B	1,250,000 B
7.15(b)	2,500,000 B	1,250,000 B

(d) Buffer Size Configuration

Buffer Splitting	**Smart Fast Ret.**	
	disabled	enabled
disabled, SACK only	× (4)	+ (3)
disabled, **NR-SACK**	△ (2)	○ (1)
enabled, **NR-SACK**	▽ (6)	◇ (5)

(e) Features Configuration

Figure 7.15: Throughput for Unordered Delivery over Paths with Dissimilar Delays

In order to examine the impact of dissimilar delays, the delay of Path #2 has been varied from δ_2=1 ms to δ_2=40 ms, while keeping the delay of Path #1 fixed at δ_1=1 ms. Figure 7.15 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios. Subfigure 7.15(d) provides the buffer size configurations; Subfigure 7.15(e) contains the used combination of mechanisms (i.e. both-side buffer splitting based on outstanding bytes, NR-SACK and smart fast retransmission) and their corresponding point symbols and curve numbers. Note, that the buffer sizes are larger in comparison to the bandwidth and bit error rate scenarios of Subsection 7.7.1 and Subsection 7.7.2, due to the higher bandwidth-RTT product (see Subsubsection 2.9.2.3) of Path #2 for higher delays.

As expected, the performance for buffer splitting as well as NR-SACKs turned off is not very good, regardless of smart fast retransmission usage (curves 3 and 4; the curves are superimposed). The achieved application payload throughput remains quite constant at about 155 Mbit/s. Clearly, this is better than the ideal-case non-CMT performance of about 95 Mbit/s. That is, the mechanisms of CMT-SCTP to handle delay differences among the paths (split fast retransmission as described in Subsection 4.2.2, congestion window update for CMT as described in Subsection 4.2.3) avoid extreme performance effects. Note, that the buffer size settings in this scenario even do not reach the full performance for a similar path setup (i.e. δ_2=1 ms), due to GapAck-induced send buffer blocking. This underutilisation of the paths (by short transmission stalls) results in no visible impact of the congestion window reductions caused by unnecessary fast retransmissions, i.e. smart fast retransmission (curve 3) shows no difference to the configuration without this feature (curve 4).

The issue of GapAck-induced send buffer blocking is avoided by turning on NR-SACKs (curves 1 and 2 as well as curves 5 and 6), which achieves a significant performance improvement – particularly for δ_2<40 ms. The usage of buffer splitting (curves 5 and 6) in comparison to buffer splitting turned off (curves 1 and 2) does not have a significant impact on the performance. That is, the curves 1 and 5 as well as 2 and 6 are superimposed. As explained above, the mechanisms already provided by CMT-SCTP handle delay differences very well.

The usage of smart fast retransmission (i.e. curves 1 and 5, drawn as solid lines) significantly improves the performance for higher settings of the Path #2 delay δ_2. At e.g. δ_2=40 ms, a throughput of 175 Mbit/s is achieved in comparison to only about 155 Mbit/s for turning this feature off (curves 2 and 6, drawn as dashed lines). As explained in Section 3.8, a fast retransmission halves the congestion window. Since each growth step – by just one path MTU – in congestion avoidance mode requires the successful acknowledgement of a complete congestion window of data, the congestion window for a high-delay path returns to its original size very slowly. Therefore, avoiding any unnecessary fast retransmission becomes important, which makes smart fast retransmission a necessary feature to achieve a good performance on dissimilar paths with high delay differences.

7.7.4 Summary

In summary, the parameter study for unordered delivery has shown that efficient CMT-SCTP-based load sharing over dissimilar paths is possible over a wide parameter range, if applying the following mechanisms in combination:

- Buffer splitting based on outstanding bytes (see Section 7.6) to avoid transmission-induced send buffer blocking as well as advertised-window-induced receive buffer blocking,

- the NR-SACK extension (see Subsection 3.11.5 and [NEY+08]) to prevent GapAck-induced send buffer blocking,

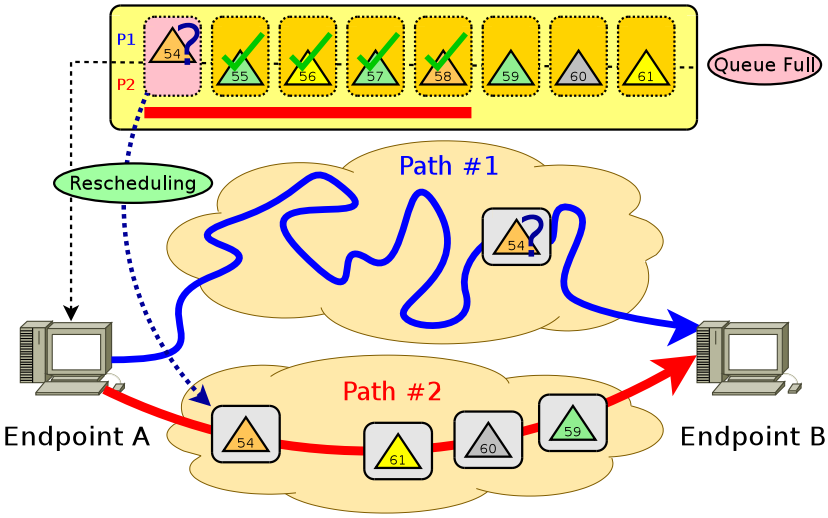


Figure 7.16: The Principle of Chunk Rescheduling

- Smart fast retransmission (see Subsection 7.7.3) to avoid spurious bursts of fast retransmissions for dissimilar delays as well as
- the Packet drop reporting extension (see Subsection 3.11.8 and [RTR09]) to efficiently handle packet corruption.

7.8 Chunk Rescheduling

Based on the solved challenges of unordered delivery, it is straightforward to continue with data transfer using ordered delivery. As explained in Subsubsection 7.5.2.2, reordering-induced receive buffer blocking leads to missing DATA chunks blocking a CumAck – and therefore preventing a removal of a possibly large set of other DATA chunks – in the receive buffer. Since for ordered delivery all DATA chunk payload must be delivered to the upper layer in sequence, a forwarding of received DATA chunks – as it can be performed for unordered delivery – is no possibility here.

An example is provided in Figure 7.16. Here, the DATA chunks #54 to #61 have been sent by Endpoint A. The destination, Endpoint B, has already GapAck'ed the TSNs #55 to #58. However, in order to perform a CumAck for TSN #58, the DATA chunk #54 – which has been the only chunk sent via Path #1 – is still missing yet. Without this chunk, the already received data cannot be provided to the upper layer in sequence. Since the send buffer is full, no further new DATA chunks can be sent into the network. That is, the transmission will be stalled until a successful acknowledgement of DATA chunk #54 (by a CumAck for TSN #58). In particular, the DATA chunks on Path #2 are blocked by the missing chunk on Path #1.

As a solution to the reordering-induced receive buffer blocking, the approach of *Chunk Rescheduling* has been developed as part of this thesis and published in [DBRT10]. In short, if one path occupies a significant fraction of the buffer space by queued – but not outstanding – DATA chunks, the first DATA chunk which is still outstanding and therefore contributing to the problem is scheduled for retransmission on this path. This procedure is denoted as *Rescheduling*. In the example above, DATA chunk #54 is rescheduled and retransmitted on Path #2, which would otherwise be under-

utilised anyway. When either the original chunk or the retransmission arrives at Endpoint B, the receiver instance can generate a CumAck for TSN #58 and therewith solve the blockade.

In order to detect whether one path P of the n paths to a peer side is blocking the send buffer, the so-called *Sender Blocking Fraction* is calculated as follows:

$$\mathrm{SenderBlockingFraction}_P = \begin{cases} \frac{\mathrm{Queued}_P - \mathrm{Outstanding}_P}{B^{\mathrm{Sender}}} & \text{(No Buffer Splitting)} \\ \frac{\mathrm{Queued}_P - \mathrm{Outstanding}_P}{B^{\mathrm{Sender}}/n} & \text{(Buffer Splitting)} \end{cases}.$$

Here, Queued_P denotes the number of queued bytes on path P. When applying buffer splitting, each of the n paths should get its $\frac{1}{n}$-th share of the send buffer.

Similar to the sender blocking fraction, the so-called *Receiver Blocking Fraction* can be calculated by estimating the size of the receive buffer from the peer receiver window (similar to receive buffer splitting, as described in Subsubsection 7.6.1.1):

$$\mathrm{ReceiverBlockingFraction}_P = \begin{cases} \frac{\mathrm{Queued}_P - \mathrm{Outstanding}_P}{\mathrm{PeerReceiverWindow} + \sum_p \mathrm{Queued}_p} & \text{(No Buffer Splitting)} \\ \frac{\mathrm{Queued}_P - \mathrm{Outstanding}_P}{\left(\mathrm{PeerReceiverWindow} + \sum_p \mathrm{Queued}_p\right)/n} & \text{(Buffer Splitting)} \end{cases}.$$

Note, that the calculation must add the queued bytes to the peer receiver window here, since the receive buffer will not only be utilised by DATA chunks that are still in-flight. It actually *is* utilised by the queued but already acknowledged bytes, which cannot be CumAck'ed yet. Would they have been forwarded to the upper layer, they would have been NR-SACK'ed – and would therefore also not utilise the send buffer any more.

Based on the definitions of sender blocking fraction and receiver blocking fraction of a path P, the chunk rescheduling algorithm as presented in Listing 4 can be applied:

- If the path P is in fast recovery mode (as described in Subsubsection 2.11.2.2) or already blocked (to be explained below), nothing has to be done (lines 4 to 6). Since the path at the moment has a – temporary – difficulty, it seems not to be a good idea to further add preventive retransmissions to this path.

- Otherwise, the sender blocking fraction and receiver blocking fraction are obtained (lines 9 and 10).

- If either one blocking fraction value exceeds the threshold $\mathrm{ChunkReschedulingThreshold}$, the chunk rescheduling functionality is triggered (lines 12 and 13). A setting of 50% for this parameter has shown to be useful, since it does not trigger the chunk rescheduling too early, but still leaves some time for the preventive retransmission to succeed.

- Then, it is checked whether the DATA chunk with the lowest TSN t in the send buffer satisfies the following conditions (lines 16 to 20):

  1. It is still outstanding (i.e. it had been transmitted, but it has not been acknowledged yet),
  2. It has not been moved from another path to P by a prior chunk rescheduling invocation (in order to avoid oscillation),
  3. It has not been transmitted on path P *or* its transmission on P has occurred at least two times the SRTT (i.e. the smoothed round-trip time as defined in Subsection 2.11.4) of path P ago (i.e. it has quite likely been lost).

Listing 4 The Chunk Rescheduling Algorithm

```
void chunkReschedulingControl(Path* P)
{
   // ====== Nothing to do when P is in Fast Recovery or blocked ========
   if( (P->IsInFastRecovery) || (P->IsBlocked) ) {
      return;
   }

   // ====== Calculate Blocking Fractions ================================
   Calculate SenderBlockingFraction[P];
   Calculate ReceiverBlockingFraction[P];

   if( (SenderBlockingFraction[P] >= ChunkReschedulingThreshold) ||
       (ReceiverBlockingFraction[P] >= ChunkReschedulingThreshold) ) {
      // ====== Path P blocks significant fraction of a buffer ==========

      t = getLowestTSNinSendBuffer();
      if((Chunk[t]->IsOutstanding == true) &&
         (Chunk[t]->HasBeenMoved == false) &&
         ( (Chunk[t]->LastPath != P) ||
           (Chunk[t]->LastSendTime + (2 * P->SRTT) < getCurrentTime())) )
      {
         // ======  Reschedule chunk ======================================
         moveChunkToPath(Chunk[t], P);

         if((Chunk[t]->LastPath != P) &&
            (Chunk[t]->LastPath->IsBlocked == true)) {
            // ====== Path is already blocked ============================
            // Nothing to do
         }
         else if((Chunk[t]->LastPath != P) &&
                 (Chunk[t]->LastPath->IsInFastRecovery == true)) {
            // ====== Path is in Fast Recovery => block it ==============
            Chunk[t]->LastPath->IsBlocked  = true;
            Chunk[t]->LastPath->BlockUntil = getCurrentTime() +
                                             Chunk[t]->LastPath->SRTT;
            // No more *new* chunks on this path, for one RTT!
         }
         else {
            // ====== Path is not in Fast Recovery yet ==================
            startFastRecovery(Chunk[t]->LastPath);
         }
      }
   }
}
```

- If such a DATA chunk t has been found, it is rescheduled on path P for preventive retransmission (line 23). Since a quick acknowledgement of the missing DATA chunk t is highly important for the transmission progress, its I-Bit is set if the SACK immediately extension (as explained in Subsection 3.11.6) is supported. That is, the receiver should send an acknowledgement for this DATA chunk immediately, without applying delayed acknowledgement.
- Then, the previous transmission path Q of the rescheduled DATA chunk t is processed as follows (lines 25 to 41):
  - If path Q, $Q \neq P$ is already blocked (to be described below), nothing is done.
  - Else, if path Q, $Q \neq P$ is in fast recovery mode (see Subsubsection 2.11.2.2), it will be blocked for new DATA chunk transmissions for one SRTT of path Q (see also Subsection 2.11.4). That is, it will not be used for any *new* data, but retransmissions and control chunks are – of course – still sent on the blocked path Q. The reason is to let the path recover from possible congestion issues.
  - Else, since path Q ($Q = P$ or $Q \neq P$) is not in fast recovery yet, a fast recovery is started. This in particular also means to decrease the congestion window of path Q (see Subsubsection 2.11.2.2).

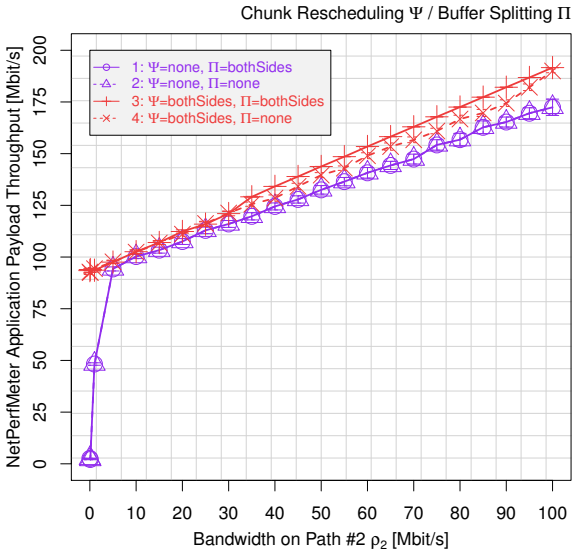
Note, that from the congestion control perspective, the previous path Q of a DATA chunk having been rescheduled is handled like having lost this chunk due to congestion. Rescheduled retransmissions are handled like regular retransmissions, i.e. a DATA chunk is only sent into the network when the congestion window of the corresponding path allows its transmission. Furthermore, since it is not known for sure on which path the retransmitted DATA chunk finally gets acknowledged (the original transmission may have been successful), a rescheduled DATA chunk will not be counted as *new* acknowledgement during the congestion window update procedure explained in Subsection 4.2.3. That is, it will *not* contribute to a congestion window growth. Therefore, chunk rescheduling is not more aggressive than regular CMT-SCTP.

7.9 Ordered Delivery

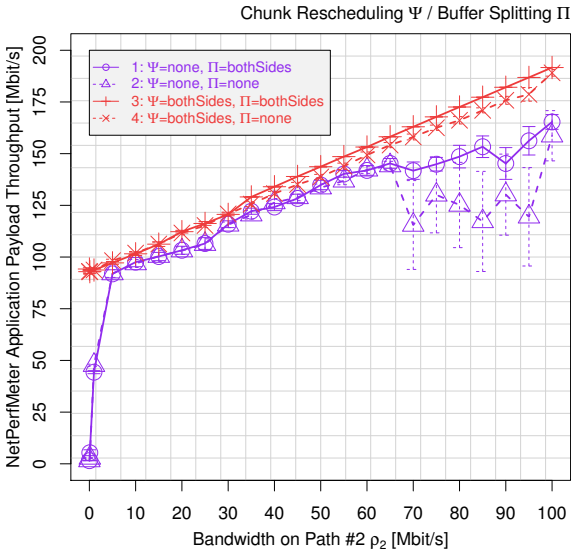
In order to show how to provide efficient ordered delivery with chunk rescheduling, a simulative parameter study has been performed. Its structure is similar to the unordered delivery case described in Section 7.7, i.e. separated into dissimilarity introduced by varying bandwidth, bit error rate and delay. The initial version of this parameter study has also been published in [DBRT10].

7.9.1 Dissimilar Bandwidths

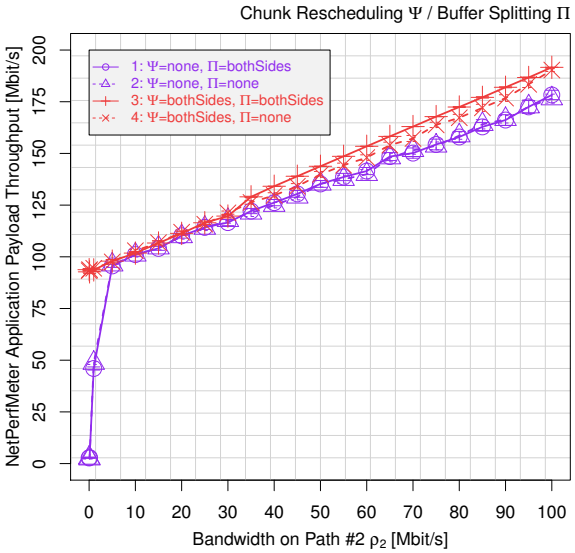
In order to demonstrate the influence of bandwidth dissimilarity on ordered delivery, the bandwidth of Path #2 has been varied from ρ_2=10 Kbit/s (i.e. 0.01 Mbit/s) to ρ_2=100 Mbit/s, while keeping the bandwidth of Path #1 fixed at ρ_1=100 Mbit/s. Figure 7.17 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios. Subfigure 7.17(d) shows the buffer size configurations, Subfigure 7.17(e) contains the used combinations of mechanisms (chunk rescheduling and buffer splitting based on outstanding bytes) and their corresponding point symbols and curve numbers. That is, while the throughput of 2×100 Mbit/s is $2.5 * 10^7$ bytes/s (i.e. more than 23.8 MiB/s), the buffer sizes are by one order of magnitude smaller, as motivated in Section 7.4.



(a) Send Buffer Size < Receive Buffer Size



(b) Send Buffer Size > Receive Buffer Size



(c) Send Buffer Size = Receive Buffer Size

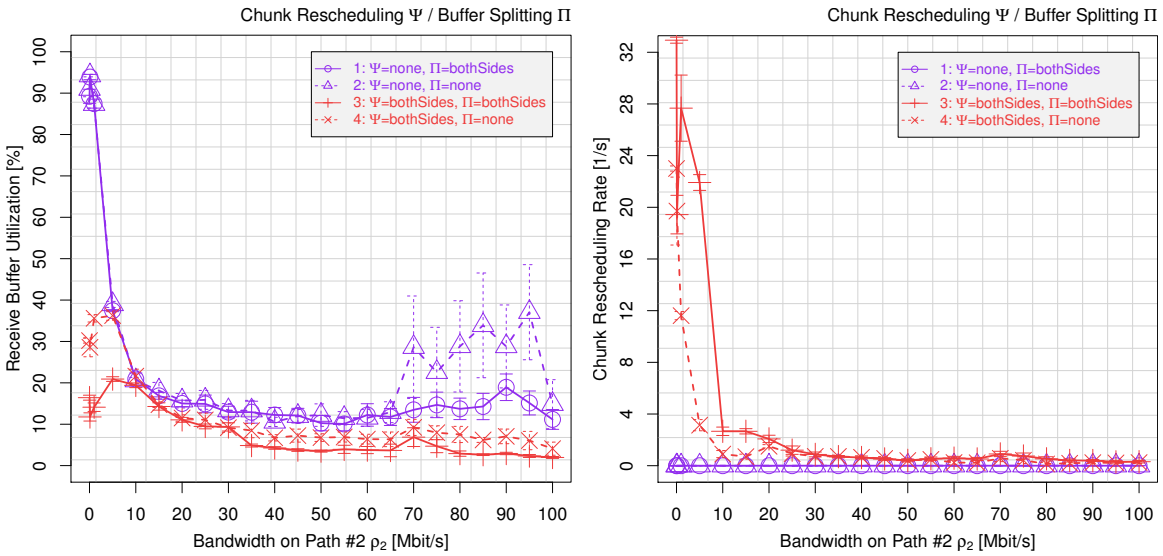
Figure	Send Buffer	Receive Buffer
7.17(a)	2,500,000 B	5,000,000 B
7.17(c)	2,500,000 B	2,500,000 B
7.17(b)	5,000,000 B	2,500,000 B
7.18		

(d) Buffer Size Configuration

Chunk Rescheduling	**Buffer Splitting**	
	disabled	enabled
disabled	△ (2)	○ (1)
enabled	× (4)	+ (3)

(e) Features Configuration

Figure 7.17: Throughput for Ordered Delivery over Paths with Dissimilar Bandwidths



(a) Receive Buffer Utilisation

(b) Chunk Rescheduling Rate

Figure 7.18: Dissimilar Bandwidths in the Send Buffer Size > Receive Buffer Size Scenario

First, if chunk rescheduling is turned off (curves 1 and 2), the usage of CMT-SCTP clearly improves the ordered delivery performance in comparison to a non-CMT configuration. For example, at ρ_2=50 Mbit/s, the achieved application payload throughput is about 130 Mbit/s in all three cases of send buffer size to receive buffer size ratios. However, this is not the full expected throughput. Furthermore, a too-dissimilar setup – e.g. $\rho_2 \leq 1$ Mbit/s – leads to an extreme performance drop.

The reason for this performance drop is reordering-induced receive buffer blocking (as described in Subsubsection 7.5.2.2), which means that missing DATA chunks prevent a CumAck of already-received chunks in the receive buffer. In consequence, the lack of a CumAck causes the issue of advertised-window-induced send buffer blocking (as described in Subsubsection 7.5.2.1). Since this effect is clearly the strongest when the receive buffer is smaller than the send buffer, this case shows the highest performance reduction (see Subfigure 7.17(b)). This reduction can particularly also be observed in "relatively similar" bandwidth cases (here: $\rho_2 \geq 70$ Mbit/s to ρ_2=100 Mbit/s).

In order to further explain these effects, Figure 7.18 presents a more detailed look at the buffer behaviour: the average receive buffer utilisation is shown in Subfigure 7.18(a). For buffer splitting turned off (curve 2), there are local peaks for the receive buffer utilisation at $\rho_2 \geq 70$ Mbit/s, due to advertised-window-induced receive buffer blocking. Here, the two – relatively fast – paths easily reduce the peer receiver window (see Subsection 2.10.1) down to zero, frequently leading to a temporarily stalled transmission. Buffer splitting (curve 1) solves the problem of advertised-window-induced receive buffer blocking, but the issue of high reordering-induced buffer blocking due to the high path bandwidths remains.

Furthermore, the receive buffer utilisation quickly rises to slightly below 100% for a highly dissimilar setup at $\rho_2 \leq 5$ Mbit/s if buffer splitting is turned off (curve 2). The reason is the issue of buffer bloat (see also Subsection 7.6.3), which causes an overly long delay for the DATA chunks on Path #2 at small bandwidth settings ρ_2. Buffer splitting (curve 1) cannot reduce the problem, since it does not prevent reordering-induced receive buffer blocking.

Turning on chunk rescheduling (curves 3 and 4) solves the problems caused by reordering-induced

receive buffer blocking. As shown in Figure 7.17, the application payload throughput is significantly improved in all three cases. In particular, it now reaches the theoretically expected value in most cases and there is no performance drop for highly dissimilar scenarios of $\rho_2 \leq 1$ Mbit/s. Chunk rescheduling even achieves an acceptable performance at ρ_2=10 Kbit/s (i.e. 0.01 Mbit/s), which is by four orders of magnitude smaller than the throughput of ρ_1=100 Mbit/s of Path #1. For comparison, without chunk rescheduling, the achieved throughput here is almost zero Mbit/s (see curves 1 and 2). While turning off buffer splitting still improves the performance (curve 4), the best performance is reached when applying chunk rescheduling in combination with buffer splitting (curve 3).

Concerning overhead, chunk rescheduling is also reasonably inexpensive, as shown by the chunk rescheduling rate depicted in Subfigure 7.18(b) for the most challenging scenario of the send buffer larger than the receive buffer. For $\rho_2 \leq 30$ Mbit/s, the chunk rescheduling rate (i.e. the number of rescheduled chunks per second; see lines 21 to 42 of Listing 4) is less than 1 chunk/s (i.e. one full-sized packet/s). For comparison, the number of 1,500 bytes packets at a bandwidth of 100 Mbit/s is 8,333.$\overline{3}$ packets/s. Even in the highly dissimilar scenarios at $\rho_2 \leq 1$ Mbit/s, the chunk rescheduling rate is only about 30 chunks/s. That is, the number of *possibly* duplicated DATA chunks in order to significantly improve the performance remains small.

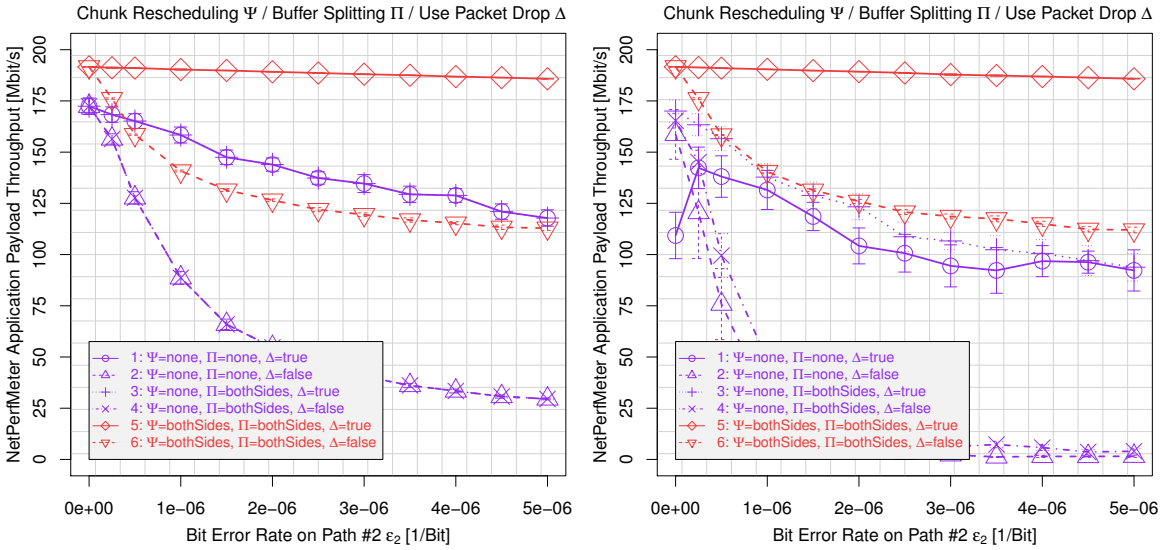
A further notable observation on the chunk rescheduling rate is the difference between buffer splitting enabled (curve 3) and buffer splitting disabled (curve 4) for a Path #2 bandwidth of $\rho_2 \leq 30$ Mbit/s. While there is no significant application payload throughput difference between these two cases (see Figure 7.17), the overhead is higher when buffer splitting is applied. Here, the buffer splitting rules (see Section 7.6) sometimes delay the transmission of rescheduled chunks, since the limit for the outstanding bytes of a path is reached. Then, a further chunk rescheduling (for the next chunk in the send buffer) may be triggered, leading to the increased overhead without a further performance improvement. As future work, a heuristic may be developed to prevent this overhead.

7.9.2 Dissimilar Bit Error Rates

For presenting the impact of bit error rate dissimilarity, the bit error rate of Path #2 has been varied from ϵ_2=0 to $\epsilon_2 = 5 * 10^{-6}$ errors/bit, while keeping Path #1 error-free (i.e. ϵ_1=0). Figure 7.19 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios; Subfigure 7.19(d) lists the buffer size configurations and Subfigure 7.19(e) shows the used combinations of mechanisms (chunk rescheduling, buffer splitting based on outstanding bytes and the packet drop reporting as described in Subsection 3.11.8) with their corresponding point symbols and curve numbers.

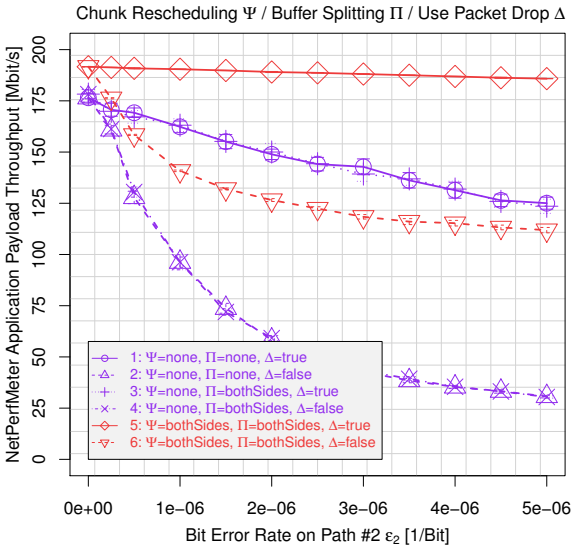
Clearly, as already observed for the dissimilar bandwidth scenario described in Subsection 7.9.1, the performance quickly decreases for an increasing bit error rate dissimilarity. At a Path #2 bit error rate of $\epsilon_2 = 5 * 10^{-6}$ errors/bit for the send buffer size $\leq$ the receive buffer size, the achieved application payload throughput is only about 30 Mbit/s if packet drop reporting is turned off (curves 2 and 4 in Subfigure 7.19(a) and Subfigure 7.19(c)) – which is approximately only one third of the non-CMT ideal-case (i.e. loss-free) performance. The scenario of send buffer size $>$ receive buffer size (see Subfigure 7.19(b)) is even worse. At the same bit error rate, the throughput drops down to almost zero Mbit/s. Obviously, the situation with support for packet drop reporting (curves 1 and 3) is significantly better, but does not solve the core problem – which is reordering-induced receive buffer blocking. Again, buffer splitting only slightly improves the situation (curves 3 and 4), since it is a countermeasure against advertised-window-induced receive buffer blocking (see Subsubsection 7.5.2.1), but not against reordering-induced receive buffer blocking (see Subsubsection 7.5.2.2).

Figure 7.20 presents a more detailed look at the buffer behaviour. The average receive buffer



(a) Send Buffer Size < Receive Buffer Size

(b) Send Buffer Size > Receive Buffer Size



(c) Send Buffer Size = Receive Buffer Size

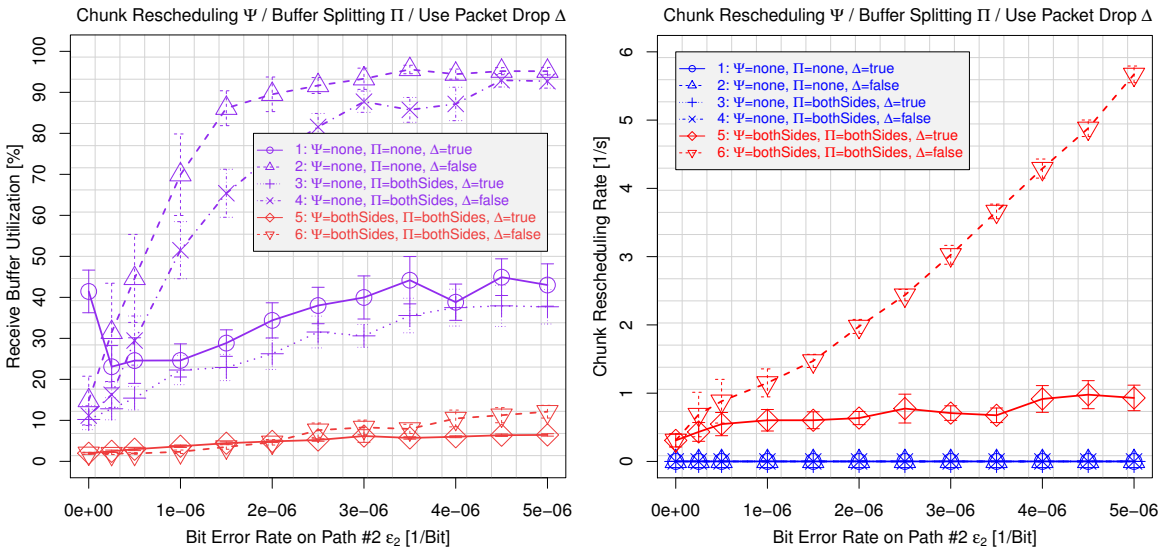
Figure	Send Buffer	Receive Buffer
7.19(a)	2,500,000 B	5,000,000 B
7.19(c)	2,500,000 B	2,500,000 B
7.19(b)	5,000,000 B	2,500,000 B
7.20		

(d) Buffer Size Configuration

Chunk Rescheduling	**Packet Drop Rep.**	
	disabled	enabled
disabled	△ (2)	○ (1)
disabled, **Buffer Splitting**	× (4)	+ (3)
enabled, **Buffer Splitting**	▽ (6)	◇ (5)

(e) Features Configuration

Figure 7.19: Throughput for Ordered Delivery over Paths with Dissimilar Bit Error Rates



(a) Receive Buffer Utilisation

(b) Chunk Rescheduling Rate

Figure 7.20: Dissimilar Bit Error Rates in the Send Buffer Size > Receive Buffer Size Scenario

utilisation depicted in Subfigure 7.20(a) makes the problem clearly visible: the higher the dissimilarity of the paths, the higher the buffer utilisation due to the reordering-induced receive buffer blocking.

An interesting observation is the negative peak of curve 1 (i.e. chunk rescheduling as well as buffer splitting turned off, but with packet drop reporting). For reasonably small error rates, the drop reports reduce the advertised-window-induced receive buffer blocking. A DATA chunk that has been reported as dropped increases the peer receiver window, since it is not outstanding any more (see also Subsection 2.10.1). Since the transmissions on the paths are handled in a round-robin manner, a non-zero peer receiver window may give a DATA chunk on the other path the chance to be sent before the retransmission of the dropped chunk. In consequence, the better scheduling of DATA chunks among the paths also reduces the reordering-induced receive buffer blocking issue and leads to some throughput performance improvements (see Subfigure 7.19(b)) in comparison to the error-free case (i.e. ϵ_2=0). However, this effect is small and highly varies, as indicated by the relatively large confidence intervals. Simply turning on buffer splitting (curve 3) is a significantly more effective countermeasure against advertised-window-induced receive buffer blocking.

Chunk rescheduling in combination with buffer splitting (curve 5) solves the problem for all three send buffer size to receive buffer size ratios (see Figure 7.19). While with support of packet drop reporting the achieved payload throughput at a Path #2 bit error rate of ϵ_2=$5 * 10^{-6}$ errors/bit is only slightly lower than for the error-free scenario (about 185 Mbit/s vs. 190 Mbit/s), it still achieves about 120 Mbit/s without this support (curve 6). For the whole examined bit error rate range of ϵ_2, the average receive buffer utilisation remains at about 12% or less (see Subfigure 7.20(a)).

Furthermore, the applied chunk rescheduling – while significantly improving the achieved performance – is reasonably inexpensive, as shown by the chunk rescheduling rate plot depicted in Subfigure 7.20(b): with packet drop reporting support, it is less than one rescheduled DATA chunk/s (curve 5), since a chunk having been reported as dropped is immediately retransmitted without need for rescheduling. Even without packet drop reporting, the number of rescheduled chunks/s does not exceed 6 chunks/s (curve 6).

7.9.3 Dissimilar Delays

In order to examine the impact of dissimilar delays, the delay of Path #2 has been varied from δ_2=1 ms to δ_2=40 ms, while keeping the delay of Path #1 fixed at δ_1=1 ms. Figure 7.21 presents the resulting application payload throughput for the three possibilities of send buffer size to receive buffer size ratios. The buffer size configurations are shown in Subfigure 7.21(d), the used combinations of mechanisms (chunk rescheduling and buffer splitting based on outstanding bytes) can be found in Subfigure 7.21(e).

As already observed for the unordered delivery scenario in Subsection 7.7.3, CMT-SCTP provides mechanisms to cope with some delay differences. The problem for ordered delivery arises when one path loses a DATA chunk that is needed for a CumAck. In consequence, this may lead to a filled send or receive buffer and a stall of the transmission on all paths until the missing chunk either gets acknowledged or successfully retransmitted.

Without chunk rescheduling (curves 1 and 2), the achieved application payload throughput performance slightly reduces from around 170 Mbit/s at a Path #2 delay of δ_2=1 ms to 160 Mbit/s for δ_2=40 ms (in the scenarios of send buffer size $\leq$ receive buffer size). The scenario of send buffer size $>$ receive buffer size is – as already observed for the bandwidth and bit error rate dissimilarity scenarios in Subsection 7.9.1 and Subsection 7.9.2 – slightly more critical.

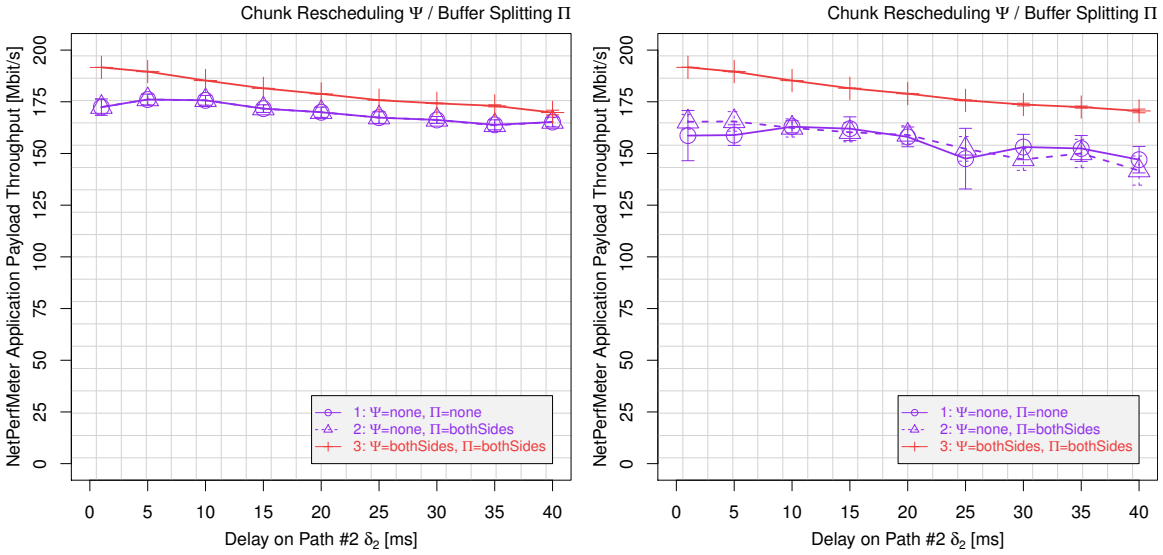
The problem caused by the reordering-induced receive buffer blocking is solved by applying chunk rescheduling in combination with buffer splitting (curve 3), which then achieves the expected throughput of about 190 Mbit/s in the similar setup (i.e. δ_2=1 ms) and still about 175 Mbit/s for a Path #2 delay of δ_2=40 ms.

Figure 7.22 presents a more detailed look at the buffer behaviour; the average receive buffer utilisation is depicted in Subfigure 7.22(a). The results here are interesting in comparison to the results of the bandwidth and bit error rate dissimilarity scenarios in Subsection 7.9.1 and Subsection 7.9.2. Even without chunk rescheduling, the average receive buffer utilisation is up to at most 20% – even in the highly dissimilar setup at δ_2=40 ms. For delay dissimilarity, the problem of reordering-induced receive buffer blocking occurs infrequently, when a temporary congestion window reduction (due to the AIMD behaviour, see Subsection 2.11.2) leads to a short stall of the transmission. After recovering from this temporary problem, the following CumAck solves the blockade. Due to the infrequent occurrence of these stalls, the overall application payload throughput reduction remains small, but chunk rescheduling can ensure to solve these situations more quickly and gain an even better performance.

Again, this performance improvement is very inexpensive, as shown by the chunk rescheduling rate in Subfigure 7.22(b) of only about 0.5 to 2 DATA chunks/s. Note the large confidence intervals. They are caused by the quite rare occurrence of a congestion window reduction for a higher delay, since it takes one RTT to rise the congestion window by one MTU in congestion avoidance mode (see Section 3.8). However, when there is a drop of the congestion window, chances are good that chunk rescheduling is triggered multiple times in sequence. Due to the non-determinism of congestion window reductions caused by the RED queues, the total number of reschedules per run varies.

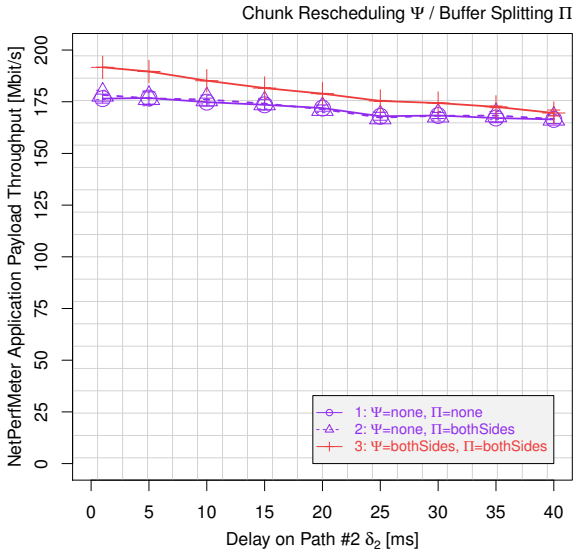
7.9.4 The Influence of the Burst Mitigation Variant

When transferring DATA chunks with ordered delivery using size-limited buffers, short pauses of the transmission on a path occur sometimes, due to reordering-induced receive buffer blocking, as observed in the delay dissimilarity scenario described in Subsection 7.9.3.



(a) Send Buffer Size < Receive Buffer Size

(b) Send Buffer Size > Receive Buffer Size



(c) Send Buffer Size = Receive Buffer Size

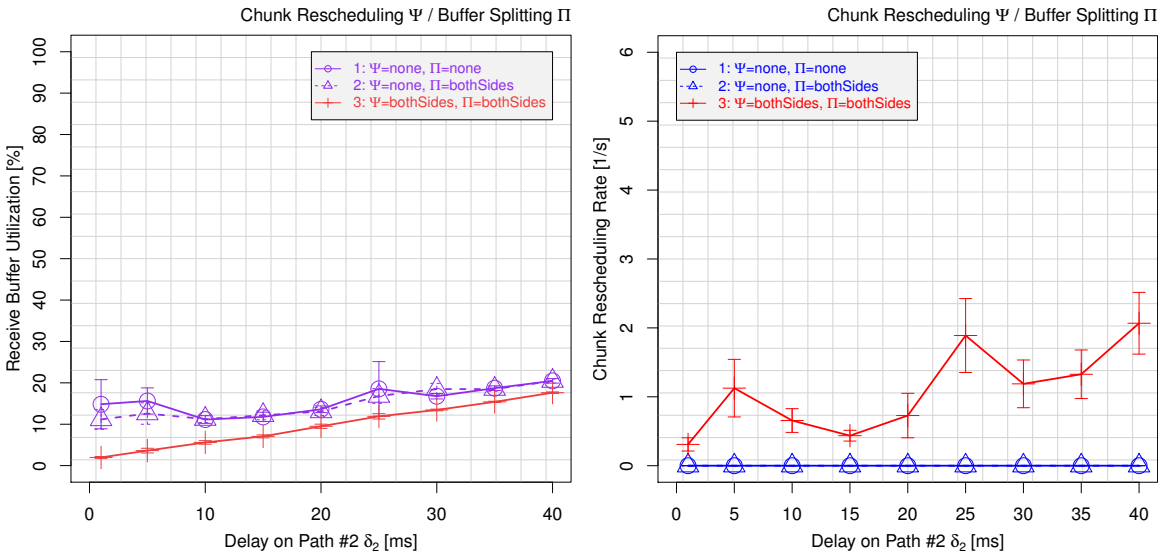
Figure	Send Buffer	Receive Buffer
7.21(a)	2,500,000 B	5,000,000 B
7.21(c)	2,500,000 B	2,500,000 B
7.21(b)	5,000,000 B	2,500,000 B
7.22		

(d) Buffer Size Configuration

Chunk Rescheduling	**Buffer Splitting**	
	disabled	enabled
disabled	○ (1)	△ (2)
enabled		+ (3)

(e) Features Configuration

Figure 7.21: Throughput for Ordered Delivery over Paths with Dissimilar Delays



(a) Receive Buffer Utilisation

(b) Chunk Rescheduling Rate

Figure 7.22: Dissimilar Delays in the Send Buffer Size > Receive Buffer Size Scenario

7.9.4.1 The Burst Mitigation Challenge

While these pauses are usually very short (i.e. a few ms), they have an impact on the congestion window. If following the suggestion of RFC 4960 (see [Ste07, Subsection 6.1]), the mechanism of "Use It or Lose It" burst mitigation – as explained in Section 3.9 – limits the congestion window when it is not adequately utilised.

An example is illustrated in Figure 7.23: the DATA chunks #78 to #83 have been sent from Endpoint A to Endpoint B over the high-delay Path #1, while the DATA chunks #84 to #89 have been sent over the low-delay Path #2. According to the SACK generation rules defined in RFC 4960 (see [Ste07, Subsection 6.7]) and the application of delayed acknowledgement for CMT (see Subsection 4.2.4), the following events happen at the receiver:

- The DATA chunks #78 and #79 (both on Path #1) arrive. A SACK chunk is sent via the high-delay Path #1, since this is the path of the last received DATA chunk.
- The same happens after arrival of the
  - DATA chunks #84 (Path #2) and #80 (Path #1),
  - DATA chunks #85 (Path #2) and #81 (Path #1),
  - DATA chunks #86 (Path #2) and #82 (Path #1) and
  - DATA chunks #87 (Path #2) and #83 (Path #1).
- The DATA chunks #88 (Path #2) and #89 (Path #2) arrive. Now, a SACK chunk is sent via Path #2.

Since Path #2 is a low-delay path, chances are good that the last SACK chunk that has been sent via Path #2 arrives at the Endpoint A first. This chunk contains new acknowledgements for six

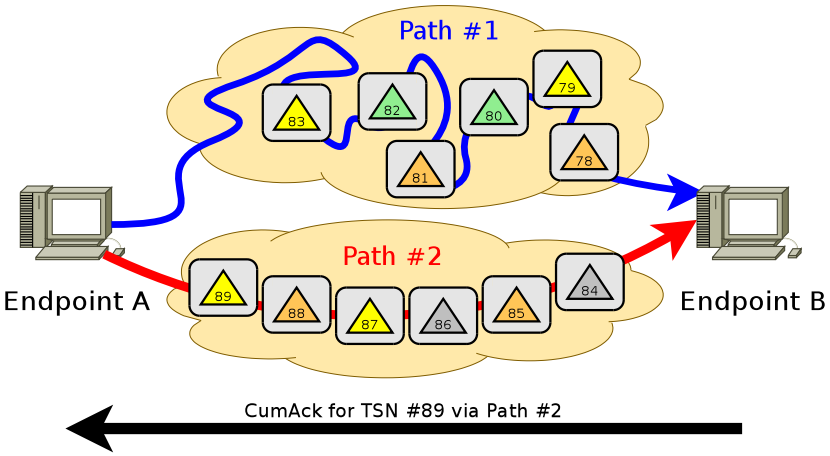


Figure 7.23: The Challenge of Burst Mitigation

DATA chunks on Path #1 as well as six DATA chunks on Path #2. That is, the corresponding number of chunks may be newly transmitted into the network. When eventually the SACKs on Path #1 arrive at Endpoint A, they are obsolete and are just ignored therefore.

In order to avoid the sender overloading the network by bursts triggered by such large acknowledgement blocks, the burst mitigation limits the number of newly sent DATA chunks. Using the approach "Use It or Lose It", the congestion window is limited so that only MaxBurst new DATA chunks (by default: 4, as defined in RFC 4960; see [Ste07, Section 15]) can be sent into the network. However, this means that the congestion window is reduced and has to grow again. Of course, if the reduced congestion window does not cover the bandwidth-RTT product, the throughput performance will suffer.

7.9.4.2 Smart SACK Path Selection

Since the key problem of the example in Figure 7.23 has been the consecutive usage of the high-delay path for SACKs (here: five times), an approach to mitigate this problem is quite straightforward: the paths of all DATA chunks received since the last SACK (or startup of the association) are remembered. Of these paths, the path having the smallest SRTT (i.e. the smoothed round-trip time as defined in Subsection 2.11.4) is used for sending out a new SACK chunk. This approach, which has been developed as part of this thesis, has been called *Smart SACK Path Selection*.

Note, that smart SACK path selection cannot introduce any unfairness in comparison to non-CMT SCTP. All paths under consideration for sending out the next SACK chunk have just (i.e. since the transmission of the last SACK chunk) received DATA chunks. Also, of course, if only DATA chunks on the high-delay path have been received, the next SACK chunk will be sent on this high-delay path.

7.9.4.3 Alternative Burst Mitigation Variants

Smart SACK path selection reduces the chance that a large burst occurs due to an awkward reception sequence of the DATA chunks. However, it cannot completely avoid it – e.g. when a sequence of DATA chunks had been received on a high-delay path, also having been acknowledged on this path, and these acknowledgements are overtaken by a SACK chunk on the low-delay path. It is therefore useful to think about alternative burst mitigation variants.

[AB05] provides a survey of burst mitigation approaches for TCP. Therefore, the adaptation of these approaches to SCTP has been considered:

Use It or Lose It This is the algorithm suggested in RFC 4960 (see [Ste07, Subsection 6.1]) and described in Section 3.9.

Congestion Window Limiting The "Congestion Window Limiting" approach is very similar to "Use It or Lose It". If the congestion window c is going to be reduced, and if the slow-start threshold s is smaller than c, s is set to c. Then, c is reduced like for "Use It or Lose It". Since the congestion control is now in slow start mode, the original value of c may be reached again more quickly than for "Use It or Lose It".

Max Burst The "Max Burst" approach simply leaves the congestion window unaffected. Instead, it just counts the number of packets and only allows MaxBurst packets at a time. When the next SACK arrives, a further burst of MaxBurst packets may be sent, etc. – as long as the congestion window allows their transmission, of course.

Aggressive Max Burst The "Aggressive Max Burst" takes obsolete (i.e. reordered) acknowledgements into account, which is useful for TCP implementations without selective acknowledgement. Since SCTP always supports this feature, this burst mitigation strategy is not useful here.

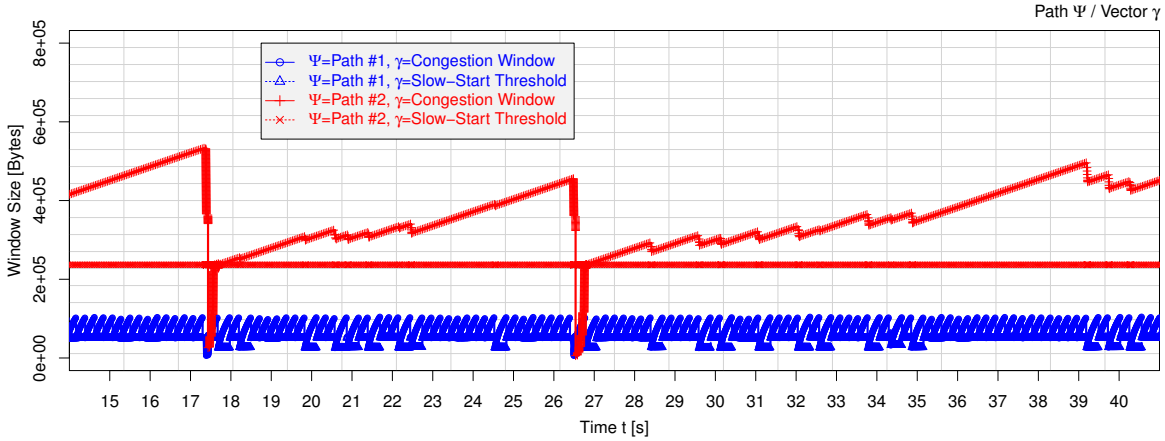
The conclusions drawn by [AB05, Section 5] show that "Use It or Lose It", "Congestion Window Limiting" and "Max Burst" are all fulfilling their task of mitigating bursts for the single-path TCP protocol, and no particular recommendation for one of these mechanisms is given. The choice of approach – for non-CMT transfer – is mainly an implementation decision, and "Use It or Lose It" is easily realisable by just a single line of C/C++ code.

7.9.4.4 Evaluation

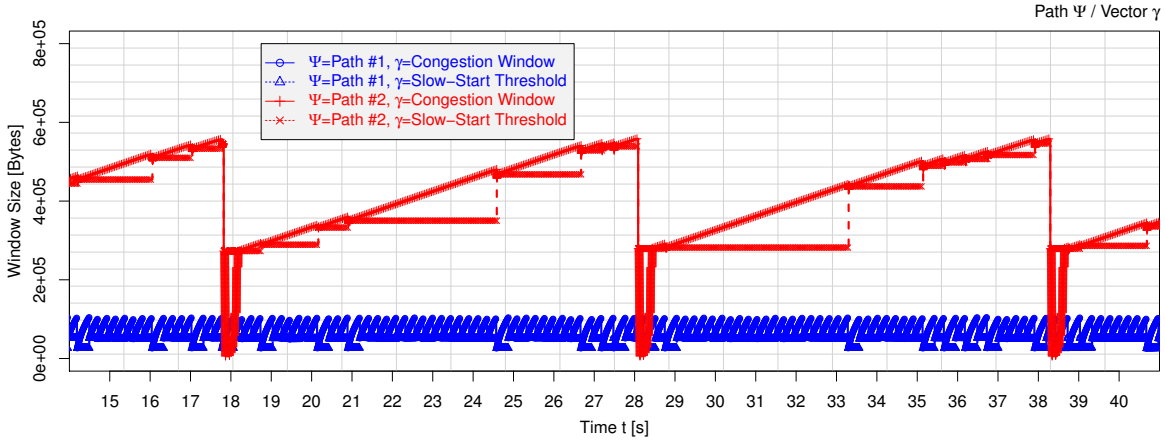
In order to demonstrate the impact of the burst mitigation variants, Figure 7.24 shows the congestion window (drawn by solid lines) and slow-start threshold (drawn by dashed lines) behaviour of the three variants for Path #1 (drawn in blue colour) and Path #2 (drawn in red colour) in the dissimilar delays scenario with chunk rescheduling from Subsection 7.9.3 at a Path #2 delay of δ_2=40 ms:

- Using "Use It or Lose It", as depicted in Subfigure 7.24(a), the problem becomes clearly visible. Path #1 shows the typical AIMD congestion control behaviour, as described in Subsection 2.11.2, i.e. the congestion window rises linearly and it is halved when the RED queue causes a loss. Unfortunately, the temporarily reduced rate of DATA chunks on Path #1 may lead to multiple SACKs in sequence using Path #2. When the congestion window of Path #1 recovers, which happens quite quickly due to a Path #1 delay of δ_1=1 ms, the new SACKs take the low-delay path and overtake the previous SACKs on Path #2. This causes the burst mitigation issue described in Subsubsection 7.9.4.1, by getting a large number of DATA chunks simultaneously acknowledged on Path #2. This results in the ragged look of the congestion window of Path #2. The resulting application payload throughput in this case is only about 160 Mbit/s.

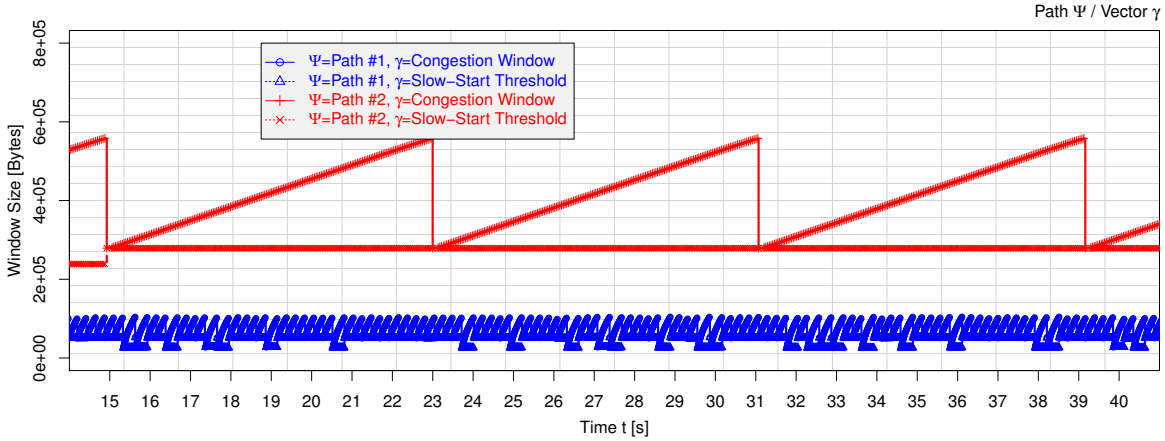
- "Congestion Window Limiting", as shown in Subfigure 7.24(b), reduces the problem by "saving" the previous value of the congestion window as slow-start threshold. This results in a faster recovery from the burst mitigation event. In this case, the resulting throughput performance is about 165 Mbit/s.



(a) Use It or Lose It



(b) Congestion Window Limiting



(c) Max Burst

Figure 7.24: The Impact of Burst Mitigation Variants on the Congestion Control Behaviour

- On the other hand, "Max Burst", as presented by Subfigure 7.24(c), lets the congestion window unaffected. That is, compared to "Use It or Lose It" or "Congestion Window Limiting", no ragging is observable. This leads to a further improvement of the application payload throughput to about 175 Mbit/s.

That is, for the best performance, it is strongly recommended to apply "Max Burst" in favour of "Use It or Lose It" for CMT-SCTP. A possible alternative is to turn off the optional burst mitigation altogether, as it is currently performed by FreeBSD kernel SCTP[5]. Since TCP does not apply burst mitigation by default, the FreeBSD developers have just decided to also turn it off for SCTP by default. However, avoiding bursts in carefully configured networks – i.e. without overly long queues causing buffer bloat – seems to be a useful feature that should not be omitted without a thorough evaluation.

7.9.5 Ongoing and Future Work

The simulations on chunk rescheduling have shown that this approach is useful to achieve a good CMT-SCTP performance for ordered delivery in the single-stream case. Scenarios using only a single stream are highly crucial, since most of the non-multimedia applications of today do not (yet) make use of multi-streaming.

As part of the ongoing work on CMT-SCTP in the DFG project this habilitation thesis has been performed within, chunk rescheduling is going to be implemented into the FreeBSD kernel SCTP implementation, in order to perform a testbed-based evaluation. Particularly, a further evaluation of the burst mitigation impact seems to be useful. These real-world evaluations are also necessary to support the contribution of chunk rescheduling into the IETF standardisation process for SCTP. Also, some additions for better support of multi-streaming and bundling (particularly, the transport of multiple DATA chunks in one packet; see also Subsubsection 2.9.3.3) are under consideration.

7.10 Predefined Stream Mapping

While chunk rescheduling copes with the problem of transferring a single stream over multiple paths, which is similar to TCP, an idea for having multiple streams is to map certain streams to certain paths.

7.10.1 Optimised Stream Scheduling for CMT-SCTP

In the example illustrated in Figure 7.25, two streams are transported via a single CMT-SCTP association over two paths. The *Stream Scheduler* of an SCTP instance decides which DATA chunk is transported over which path. RFC 4960 (see [Ste07]) makes no definition of how to apply or implement a stream scheduler. Instead, it is a decision of the SCTP implementer. A survey of scheduling algorithms is provided by [STR10], the most common approaches are round-robin (e.g. used by FreeBSD and MacOS) or first-come, first-served (e.g. used by Linux and Solaris).

As part of this thesis and in cooperation with Robin Seggelmann from the Münster University of Applied Sciences, a simple approach has been developed, which is denoted as *Predefined Stream Mapping*. It has also been published in [DSTR10]. The idea of predefined stream mapping is to apply a fixed mapping of streams to paths. That is, except in case of a path failure, the DATA chunks of the same stream always use the same path. The advantage of this approach is that delays (e.g. due to retransmissions) only affect a subset of the streams, and not the whole association. As a result, the message delay and buffer space requirements are reduced.

[5] FreeBSD 8.2 release version.

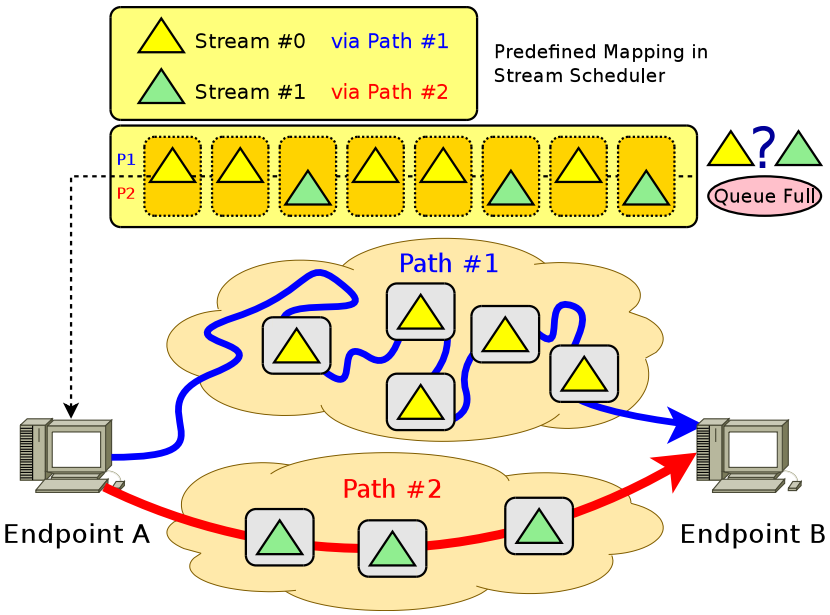


Figure 7.25: The Principle of Predefined Stream Mapping

7.10.2 Decoupled Streams

The buffers of an SCTP instance are shared among all streams. In particular, the interface between SCTP and the upper layer – as introduced in Subsection 3.12.1 and defined in [STP+11] – only provides a mechanism to trigger the generation of new messages when space in the send buffer becomes available. This is sufficient for the regular SCTP and CMT-SCTP behaviour, where streams are coupled. If two saturated streams are transmitted over the same association, each stream achieves a similar throughput. That is, the streams – regardless of the underlying paths – are similar.

On the other hand, the application of predefined stream mapping over dissimilar paths results in dissimilar streams, i.e. a stream over a high-bandwidth path may get a better throughput than a stream over a low-bandwidth path. In order to let the upper layer fill the send buffer appropriately, i.e. more messages for a high-bandwidth stream, an API extension is necessary. The proposed API extension, denoted as *Decoupled Streams*, allows the sender to query the *per-stream* send buffer utilisation. By making the Application Layer aware of the stream dissimilarity introduced by predefined stream mapping, the sender-side application can generate an appropriate amount of messages for each stream. This is particularly useful for all kinds of SCTP-based tunnelling applications, e.g. SSH channels (see Subsection 2.13.4 and [YL06]) or signalling applications like SS7 over IP networks (as described by [SMPB02]). The decoupled streams API extension, which has been developed as part of this thesis, has also been published as Internet Draft in [DSB12], and therewith contributed to the IETF standardisation process of SCTP.

7.10.3 Scenario Setup

The scenario setup for the following simulations has used the parameters described in Section 7.2, with the following modifications:

- The default QoS characteristics of the paths have been a bandwidth of 10 Mbit/s and a delay

of 10 ms without bit errors.

- The sender has transferred *two* unidirectional, saturated streams using ordered delivery over a single CMT-SCTP association to the receiver. The decoupled streams API extension, as described in Subsection 7.10.2, has been used.

- Send and receive buffers have had the same length σ.

- When applying predefined stream mapping, Stream #0 maps to Path #1 and Stream #1 maps to Path #2.

7.10.4 Dissimilar Delays

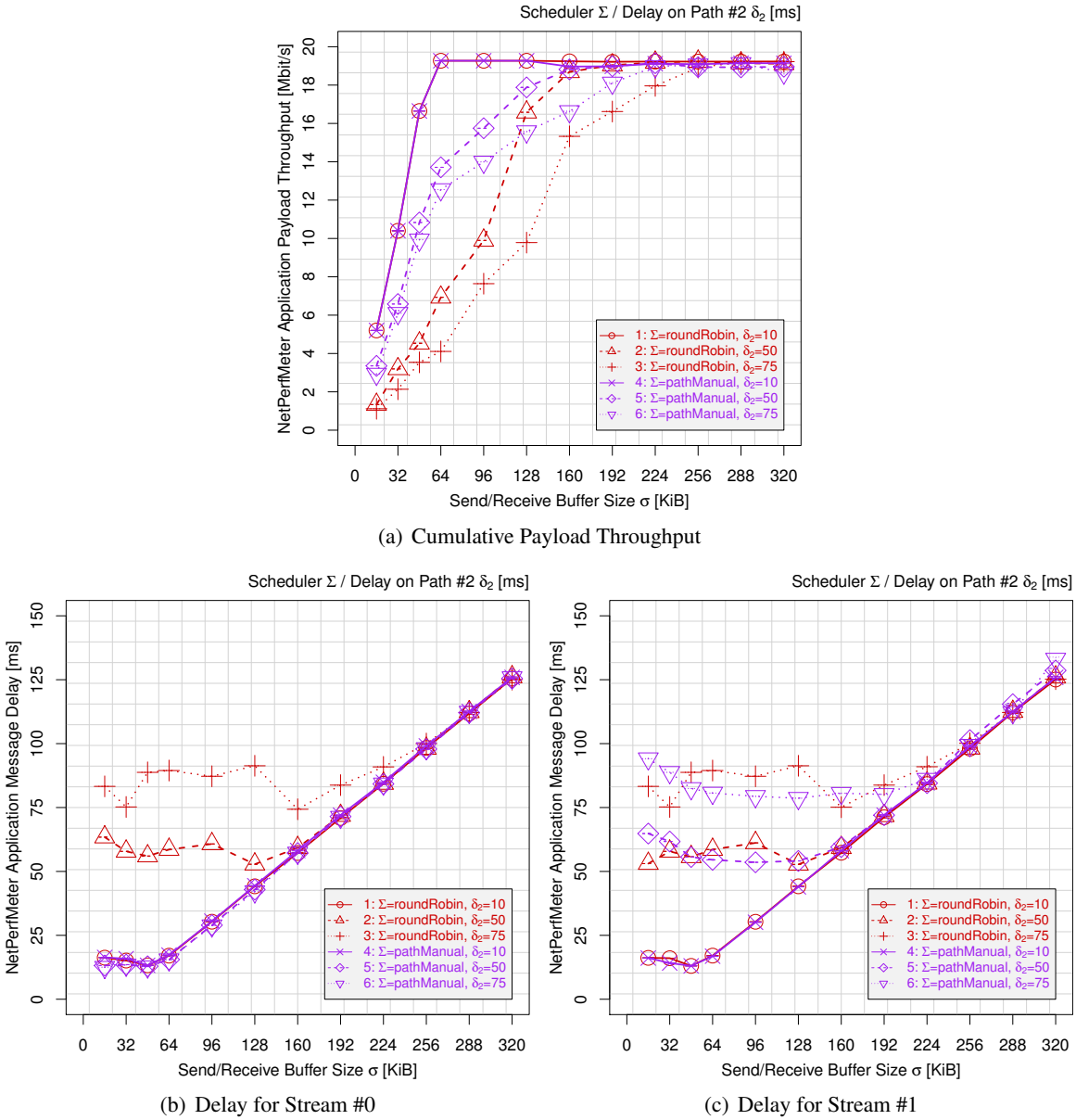
The simulation results for varying the send/receive buffer size σ from σ=16 KiB to σ=320 KiB, for using round-robin scheduling (i.e. curves 1 to 3 drawn in red colour) and predefined stream mapping (i.e. curves 4 to 6 drawn in purple colour) at Path #2 delays of δ_2=10 ms (i.e. similar paths), δ_2=50 ms and δ_2=75 ms are displayed in Figure 7.26.

Subfigure 7.26(a) presents the resulting application payload throughput. As expected from the preliminary considerations in Subsection 7.10.1, the full application payload throughput performance of about 19.2 Mbit/s is already reached for smaller settings of σ when applying the predefined stream mapping scheduling strategy (curves 4 to 6) instead of round-robin scheduling (curves 1 to 3) when the scenario becomes dissimilar (i.e. δ_2>10 ms; curves 2 and 3 as well as curves 5 and 6). Temporary delays due to losses and fast retransmissions only affect a single stream, which avoids reordering-induced receive buffer blocking for the DATA chunks of the other stream (which is transmitted over the other path). GapAck-induced send buffer blocking in this case is prevented by the usage of NR-SACKs. As expected, there is no difference between the two stream scheduler approaches in a similar path setup (i.e. δ_2=10 ms; curves 1 and 4).

The corresponding average message delay (i.e. the time from message generation at the sender-side application until reception at the peer application) is presented in Subfigure 7.26(b) (Stream #0) and Subfigure 7.26(c) (Stream #1). As expected, there is no significant message delay difference between Stream #0 and Stream #1 in the similar 10 ms/10 ms case for both scheduling approaches (curves 1 and 4). Also, the message delay linearly increases with the send/receive buffer size σ, once it is large enough to cover the bandwidth-RTT product (see Subsubsection 2.9.2.3). Any higher setting of σ cannot further improve the application payload throughput. Instead, it just increases the time the DATA chunks of a message have to wait in the send buffer until they actually get transmitted over the network. In consequence, this increases the overall message delay.

When the scenario becomes dissimilar (i.e. δ_2>10 ms), round-robin scheduling results in both streams having similar message delays, regardless of the path delay dissimilarity (curves 2 and 3) – which is the expected behaviour. On the other hand, applying predefined stream mapping (curves 5 and 6) results in dissimilar streams: Stream #0, which uses Path #1 with a delay of δ_1=10 ms, has a significantly smaller message delay than Stream #1, which uses Path #2 with a higher delay of $\delta_2 \geq$50 ms. Of course, once the send/receive buffer size σ covers the bandwidth-RTT product, the resulting throughputs of both streams become equal. Therefore, a plot for the per-stream throughputs is omitted here.

In summary, predefined stream mapping can – in comparison to round-robin scheduling – achieve the maximum throughput and a reduced delay at smaller send/receive buffer sizes.

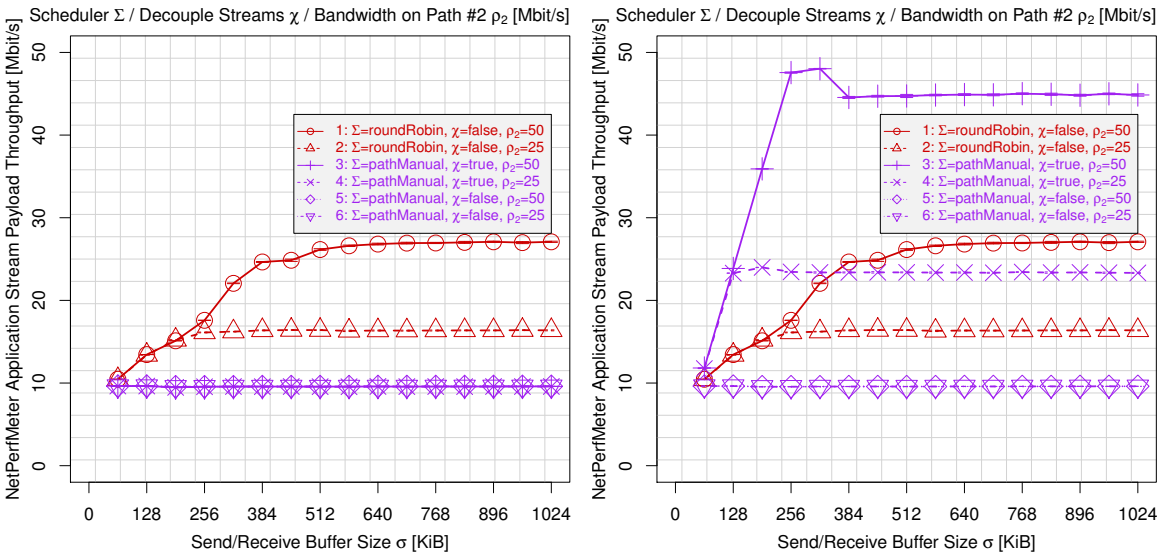


(a) Cumulative Payload Throughput

(b) Delay for Stream #0

(c) Delay for Stream #1

Figure 7.26: Using Predefined Stream Mapping for Paths with Dissimilar Delays



(a) Stream #0 (b) Stream #1

Figure 7.27: Using Predefined Stream Mapping for Paths with Dissimilar Bandwidths

7.10.5 Dissimilar Bandwidths

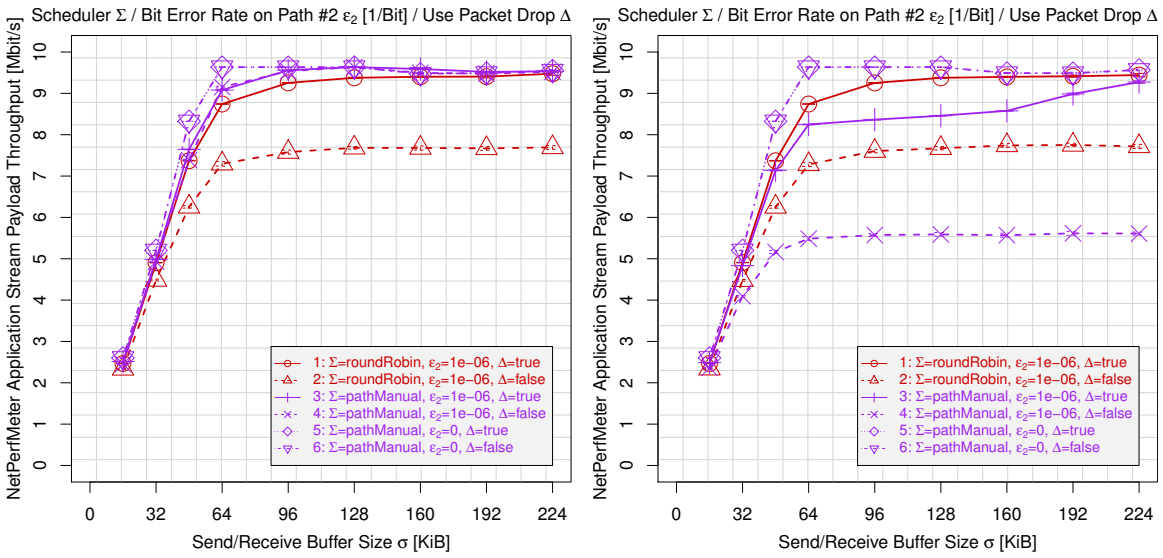
In order to demonstrate the impact of bandwidth dissimilarity, Figure 7.27 presents the application payload throughput results for varying the send/receive buffer size from σ=64 KiB to σ=1,024 KiB, for the two scheduling strategies (round-robin for curves 1 and 2 drawn in red colour; predefined stream mapping for curves 3 to 6 drawn in purple colour) at Path #2 bandwidths of ρ_2=25 Mbit/s and ρ_2=50 Mbit/s. The bandwidth of Path #1 remains fixed at ρ_1=10 Mbit/s. Curves for the similar path scenario (i.e. ρ_2=10 Mbit/s) have been omitted, since the results are obvious.

Clearly, round-robin scheduling (see curves 1 and 2) leads to an equal split of the cumulative bandwidth between both streams. That is, when the bandwidth-RTT product (see Subsubsection 2.9.2.3) is covered by the send/receive buffer size σ; e.g. for σ=768 KiB, the expected application payload throughputs of nearly $\frac{\rho_1+\rho_2}{2}$ Mbit/s per stream are reached.

When using predefined stream mapping, but without decoupled streams as explained in Subsection 7.10.2 (curves 5 and 6), the achieved application payload throughput for each stream is only the throughput of the stream on the slowest path (curves 5 and 6). That is, although Stream #1 uses the fast Path #2 with a bandwidth of $\rho_2 \geq$25 Mbit/s, its throughput remains fixed at about 9.5 Mbit/s. The problem here is that the sender-side application is not aware of the stream dissimilarities, i.e. it cannot provide enough messages to be transferred on Stream #1. Turning on the support for decoupled streams (curves 3 and 4) leads to the expected performance results. Now, the send buffer can be filled appropriately by the application. In result, CMT-SCTP is able to fully utilise the fast Path #1.

The throughput peaks at smaller settings of σ are caused by the limited buffer sizes. If the congestion window cannot grow large enough to exceed the RED queue MinTh of 30 packets (see also Subsection 2.4.2), no losses will occur. This leads to a constant size of the congestion window, instead of the typical AIMD behaviour as described in Subsection 2.11.2. Due to the high bandwidth of Stream #1 in curve 3, this effect is seen most clearly here.

Note, that predefined stream mapping already achieves this full bandwidth utilisation with a setting



(a) Stream #0 (b) Stream #1

Figure 7.28: Using Predefined Stream Mapping for Paths with Dissimilar Bit Error Rates

of $\sigma \geq$256 Kbit/s at a Path #2 bandwidth of ρ_2=50 Mbit/s, while round-robin scheduling needs a size of $\sigma \geq$768 Kbit/s. That is, predefined stream mapping can utilise the bandwidths of the two paths with smaller buffers. However, the application has to be aware of this mapping by supporting decoupled streams and it has to ensure that the send buffer gets appropriately filled with messages for each stream.

7.10.6 Dissimilar Bit Error Rates

In order to demonstrate the effects of dissimilar bit error rates, Figure 7.28 presents the per-stream application payload throughput results (Stream #0 in Subfigure 7.28(a), Stream #1 in Subfigure 7.28(b)) for varying the send/receive buffer size σ from σ=16 KiB to σ=224 KiB, for using round-robin scheduling (curves 1 and 2 drawn in red colour) and predefined stream mapping (curves 3 to 6 drawn in purple colour) at Path #2 bit error rates of ϵ_2=0 errors/bit (i.e. similar paths without errors) and $\epsilon_2 = 1 * 10^{-6}$ errors/bit, with support of the packet drop reporting extension (see Subsection 3.11.8) turned on (i.e. curves 1, 3 and 5) and off (i.e. curves 2, 4 and 6). Since the behaviour of round-robin scheduling in the similar, error-free case is obvious, the corresponding curves have been omitted in order to enhance the readability of the plots.

When using round-robin scheduling, bit errors on one path have a negative impact on the performance of *both* streams. In particular, with packet drop reporting turned off (curve 2), the achieved per-stream application payload throughput does not exceed 8 Mbit/s. Clearly, as expected from the results described in Subsection 7.7.2, turning the support for packet drop reporting on (curve 1) significantly improves the situation. However, the performance of both streams remains coupled – as observed for the bandwidth dissimilarity scenario described in Subsection 7.10.5.

Applying predefined stream mapping decouples the performances. That is, Stream #0 is now able to utilise the full bandwidth of Path #1. Regardless of the support for packet drop reporting, the application payload throughput becomes equal to the bit error-free scenario on this path (compare

curves 3 and 5 as well as curves 4 and 6 in Subfigure 7.28(a)). On the other hand, Stream #1 now suffers from the bit errors on Path #2 alone (curves 3 and 4 in Subfigure 7.28(b)).

In summary, predefined stream mapping over paths with dissimilar bit error rates can be used to improve the performance of selected streams (by mapping them to low-error paths), while the effects of losses are concentrated on the other streams. This may be useful in certain scenarios where streams have different QoS requirements, e.g. some interactive streams mixed with non-interactive ones within the same SCTP association.

7.10.7 Ongoing and Future Work

The simulations on predefined stream mapping have shown that this approach is a useful extension for CMT-SCTP in multi-streaming scenarios. However, it requires a careful configuration by the application user in order to appropriately map streams to paths. For example, for an SSH association (see Subsection 2.13.4), the mapping should use a low-delay but possibly low-bandwidth path for a stream transporting an interactive shell session, but a high-bandwidth path for a protocol tunnelling stream. The reverse choice would clearly lead to a poor user experience.

Therefore, a dynamic algorithm which automatically performs an appropriate mapping is desirable. Also, of course, an implementation and an evaluation in the testbed are part of the ongoing work on stream mapping.

7.11 Summary

In this chapter, the performance of CMT-SCTP for unordered delivery, ordered delivery and multi-streaming on dissimilar paths has been evaluated. In any realistic configuration, the sizes of send and receive buffers must be reasonably small. As a challenge in this situation, the issue of buffer blocking has been identified, which has further been categorised into four sub-problems.

For unordered delivery, the buffer splitting approach has been introduced as a solution and the effectiveness has been shown by simulations as well as testbed measurements in a real-world Internet setup. Buffer splitting decouples the send and receive buffer occupations among the paths of an association. This avoids that one path can take an overly large fraction of the send and receive buffers for its DATA chunks, leaving no more room for the other paths. Furthermore, it has been shown that buffer splitting should be combined with the non-renegable selective acknowledgements extension, in order to be able to remove successfully transmitted chunks from the send buffer as soon as possible.

As solution for ordered delivery, the approach of chunk rescheduling has been described. It realises a preventive retransmission in order to avoid stalling the data transfer by the need to wait too long for missing DATA chunks. While chunk rescheduling is quite effective, as it has been shown by simulations, the overhead for these preventive retransmissions remains reasonably small.

Finally, as an improvement for multi-streaming with ordered delivery, the approach of predefined stream mapping has been presented and evaluated. It applies a fixed mapping of streams to paths, which reduces the space requirements for send and receive buffers.

Chapter 8

Fairness on Shared Bottlenecks

This chapter introduces CMT-SCTP congestion control variants based on resource pooling, in order to address the fairness issue on shared bottlenecks, first. After that, the performance impact of these variants is analysed in a shared bottleneck scenario. Finally, their performance impact is also evaluated in disjoint path scenarios.

8.1 Introduction

A very important requirement of IETF Transport Layer protocol standardisation is *TCP-Friendliness*, as introduced in [BCC+98]. That is, a new protocol must not claim bandwidth more aggressively than a comparable TCP connection. Therefore, standard SCTP as defined in RFC 4960 (see [Ste07] and Chapter 3) only utilises its primary path and applies a congestion control similar to TCP. In order to standardise the CMT-SCTP load sharing extension, TCP-friendliness is a mandatory requirement, too.

A fundamental assumption of CMT-SCTP, as defined by [IAS06] and explained in Section 4.2, is that all paths of an association are disjoint. That is, the trails used to transfer the packets of each path have to be disjoint (as defined in Subsection 2.1.4). This makes congestion control easy: each path just has to be handled independently by using the standard SCTP congestion control mechanisms introduced in Section 3.8.

In scenarios like SS7-based telephone networks (see Subsection 3.14.1), the constraint of disjointness is quite trivial. SS7 networks are designed to provide a very high reliability, as described by [GKT00]. That is, paths must be physically redundant in order to avoid a simultaneous failure of multiple paths due to a single incident, e.g. a cut of cables due to earth works. The cables of different paths are therefore laid out at physically separate locations. Clearly, the same reliability requirements also apply when transporting SS7 signalling over IP networks by using the SCTP protocol, as described in [JRT02].

In contrast to SS7-based networks, the Internet has not been designed for high reliability. It only provides a best effort service, as described in Subsection 2.4.5. Clearly, well-defined parts of the Internet can be configured to provide disjoint paths by setting up certain routes. That is, for each path between two endpoints, the administrator may define appropriate trails to ensure disjointness. An example for this approach is the Internet-based testbed setup described in Subsection 6.5.2. However, in the general case, it is not possible to guarantee that paths between any two endpoints in the Internet are disjoint.

From the congestion control perspective, it is necessary to know whether paths are disjoint, in

order to avoid unfairness. By using plain CMT-SCTP congestion control as defined in Section 3.8, n paths of a CMT-SCTP association over the same, shared bottleneck behave like n non-CMT associations on the same bottleneck. Therefore, a CMT-SCTP association claims n-times the bandwidth share of a non-CMT association on this bottleneck. Note, that the plain CMT-SCTP congestion control only requires a logical disjointness of the paths. That is, in contrast to SS7-based networks designed for redundancy, multiple paths may physically share a single cable (e.g. by using different wavelengths, etc.). The only requirement here is that traffic on one path does not influence the QoS characteristics (i.e. bandwidth, delay, jitter and loss rate; see also Section 2.4) of another path.

In order to handle the bandwidth fairness issue on shared bottlenecks for intra-flow load sharing, the following approaches are possible:

- So-called *Bottleneck Detection* as presented by [YWY08b, YWY08a, RKT02] tries to recognise shared bottlenecks by correlating path statistics (e.g. RTTs and packet losses). While these ideas are interesting, a thorough performance evaluation in the Internet is still missing. These approaches are particularly difficult to evaluate due to the extreme heterogeneity of the Internet.

- The approach of *A-Priori Knowledge* lets the user (or administrator) decide whether paths are disjoint and load sharing should be used. This approach is applied for CMT-SCTP; it has to be enabled e.g. by a system control (as for FreeBSD described in Section 6.2) or an API option (as described in [DBA12]). While this approach is rather trivial to realise, it provides a high potential for configuration errors: CMT-SCTP could be enabled when the paths are actually shared (i.e. causing unfairness to other users), nor CMT-SCTP could be disabled even if the paths are disjoint (i.e. wasting usable bandwidth).

- The most promising approach – introduced by [WHB08] – is to assume that all paths are shared (i.e. the load sharing will be fair by default) and try to make the best of this situation (i.e. to improve the application payload throughput in comparison to non-CMT associations).

8.2 Resource Pooling

[WHB08] defines the approach of *Resource Pooling* (RP) as "making a collection of resources behave like a single pooled resource". Adapted to CMT-SCTP, this means that the set of all paths should behave like a single, high-capacity one. As performance metric for RP-enabled CMT-SCTP, the following three goals are set:

Utilisation A CMT-SCTP association should get at least as much bandwidth as a non-CMT association via the best path.

Fairness A CMT-SCTP association should not take more capacity on a shared bottleneck path than a non-CMT association via the same bottleneck.

Congestion Balancing A CMT-SCTP association should balance congestion on all of its paths. That is, it should try to shift congestion away from highly-utilised paths.

8.3 Resource-Pooling-Based Congestion Control for CMT-SCTP

The concept of RP as defined by [WHB08] is generic. However, for an evaluation of RP-based congestion control for CMT-SCTP, this concept had to be adapted to a concrete, window-based congestion

control mechanism that fits with the properties of CMT-SCTP. Three mechanisms have been realised: two RP-based congestion control variants which have been the result of observations in the testbed (see Chapter 6) and experiments with the simulation model (see Chapter 5) as well as an adaptation of the suggested congestion control for MPTCP as defined in [RHW09] to CMT-SCTP.

8.3.1 CMT/RP Congestion Control

Two variants of the simple congestion control – denoted as CMT/RP (i.e. CMT with RP) – have been developed as part of this thesis.

8.3.1.1 Version 1 – CMT/RPv1

CMT/RP version 1 – shortly denoted as CMT/RPv1 – has been the initial approach of applying RP to CMT-SCTP. It has been published in [DBPR10a] and is now also provided by FreeBSD kernel SCTP (stable release 8.2; see [DRS+11]). CMT/RPv1 assumes the slow-start threshold to be a useful metric for the stable capacity of a path. For each path P, the *Slow-Start Threshold Ratio* $\hat{s}_P$ is defined as:

$$\hat{s}_P = \frac{s_P}{\sum_i s_i}. \tag{8.1}$$

That is, $\hat{s}_P$ is the ratio between the slow-start threshold s_P on path P and the sum of the slow-start thresholds s_i over all paths.

In order to increase the congestion window c_P on α acknowledged bytes on path P in a fully-utilised congestion window, CMT/RPv1 adapts it as follows:

$$c_P = c_P + \begin{cases} \lceil \hat{s}_P * \min\{\alpha, \mathrm{MTU}_P\} \rceil & (c_P \leq s_P) \\ \lceil \hat{s}_P * \mathrm{MTU}_P \rceil & (c_P > s_P \wedge p_P \geq c_P) \end{cases}.$$

The variable p_P is the "partially acknowledged" counter for path P in congestion avoidance mode (see Subsubsection 2.11.2.1). c_P is increased according to the slow-start threshold ratio $\hat{s}_P$ of P. Note, that the ceiling function ensures that c_P is at least increased by one byte. This is necessary in order to always retain the AIMD behaviour, i.e. to try increasing the congestion window. Otherwise, for an awkward setting of $\hat{s}_P$, the congestion window could remain constant and the path would lose its capability to adapt to possible bandwidth increases.

Upon a retransmission on path P, CMT/RPv1 adapts the slow-start threshold s_P and congestion window c_P as follows:

$$\begin{aligned} s_P &= \max\left\{ \left\lceil c_P - \frac{1}{2} * \sum_i c_i \right\rceil, \lceil \hat{s} * \mathrm{MTU}_P \rceil, \mathrm{MTU}_P \right\}, \\ c_P &= \begin{cases} s_P & \text{(Fast Retransmission)} \\ \mathrm{MTU}_P & \text{(Timer-Based Retransmission)} \end{cases}. \end{aligned}$$

That is, CMT/RPv1 reduces the congestion window c_P by half of the total congestion window $\sum_i c_i$ (i.e. the sum of the congestion windows c_i on all paths) of the association, with a lower bound of MTU_P. The reason for this lower bound will be explained in Section 8.4.

8.3.1.2 Version 2 – CMT/RPv2

CMT/RPv1 assumes comparable slow-start thresholds s_i in the computation of the slow-start threshold ratio $\hat{s}_P$ (see Equation 8.1) of a path P. However, this may be difficult in case of dissimilar paths. For example, buffer bloat on one path – like in the DSL scenario described in Subsection 7.6.3 – leads to a very large congestion window and slow-start threshold at a low bandwidth, while a high-speed, low-delay path only has a small congestion window and slow-start threshold. The advanced approach CMT/RP version 2 – shortly denoted as CMT/RPv2 – overcomes the limitations of CMT/RPv1 by considering path bandwidths. This approach has been published in [DBAR11b].

In order to increase the congestion window c_P on α acknowledged bytes on path P in a fully-utilised congestion window, the so-called *Increase Factor* $\hat{i}_P$ is calculated:

$$\hat{i}_P = \frac{\frac{c_P}{\mathrm{RTT}_P}}{\sum_i \frac{c_i}{\mathrm{RTT}_i}}.$$

The increase factor represents the current bandwidth share of path P on the total bandwidth of the flow (see also Subsubsection 2.9.2.3). By using $\hat{i}$, the congestion window c_P is adapted as follows:

$$c_P = c_P + \begin{cases} \lceil \hat{i} * \min\{\alpha, \mathrm{MTU}_P\} \rceil & (c_P \leq s_P) \\ \lceil \hat{i} * \mathrm{MTU}_P \rceil & (c_P > s_P \wedge p_P \geq c_P) \end{cases}.$$

Note, that the ceiling function here also ensures at least a congestion window growth of one byte, in order to retain the AIMD behaviour.

For reducing c_P on a packet loss on path P, the *Decrease Factor* $\hat{d}_P$ is applied:

$$\hat{d}_P = \max\left\{\frac{1}{2}, \frac{1}{2} * \frac{\sum_i \frac{c_i}{\mathrm{RTT}_i}}{\frac{c_P}{\mathrm{RTT}_P}}\right\}.$$

$\hat{d}$ represents the factor by which the bandwidth of path P should be reduced in order to halve the total bandwidth of the flow. For example, two paths P_1 (10 Mbit/s) and P_2 (2 Mbit/s) lead to a total bandwidth of 12 Mbit/s. A loss on P_1 leads to $\hat{d}_1 = \frac{1}{2} * \frac{12}{10} = 0.6$; a loss on P_2 to $\hat{d}_2 = \frac{1}{2} * \frac{12}{2} = 3.0$. Note, that $\hat{d}_P \geq \frac{1}{2}$, i.e. the following reduction of slow-start threshold and congestion window should be at least as strong as for standard SCTP (see Section 3.8). By using $\hat{d}_P$, the slow-start threshold s_P and congestion window c_P are adapted as follows:

$$\begin{aligned} s_P &= \max\left\{c_P - \lceil \hat{d}_P * c_P \rceil, \mathrm{MTU}_P\right\}, \\ c_P &= \begin{cases} s_P & \text{(Fast Retransmission)} \\ \mathrm{MTU}_P & \text{(Timer-Based Retransmission)} \end{cases}. \end{aligned}$$

That is, the new setting of c_P tries to halve the total bandwidth, with a lower bound of MTU_P (to be explained in Section 8.4).

8.3.2 MPTCP-Like Congestion Control

Like CMT/RP, the congestion control of MPTCP – as proposed by [RHW09] and published as Internet Draft in [RHW11] – also applies RP to ensure fairness. However, the congestion control behaviour is different: while CMT/RP tries to halve the *total* congestion window/total bandwidth on a packet

loss on path P, MPTCP congestion control behaves exactly like standard TCP or SCTP by only halving the *path* congestion window c_P (see Section 3.8 for the formula). Since this behaviour alone would cause unfairness, the increase behaviour has to be adapted. Therefore, MPTCP uses the idea of controlling engineering: increase and decrease of c_P have to be brought into equilibrium by adapting the congestion window growth by a per-flow *Aggressiveness Factor* $\hat{a}$.

Since the MPTCP congestion control introduced in [RHW09] is based on packets instead of bytes and SCTP instead of TCP is used, the MPTCP congestion control had to be ported accordingly as part of this thesis. That is, on α acknowledged bytes on path P in a fully-utilised congestion window, the *MPTCP-Like* congestion control for CMT-SCTP adapts c_P as follows:

$$c_P = c_P + \begin{cases} \min\left\{\left\lceil \frac{c_P * \hat{a} * \min\{\alpha, \mathrm{MTU}_P\}}{\sum_i c_i} \right\rceil, \min\{\alpha, \mathrm{MTU}_P\}\right\} & (c_P \leq s_P) \\ \min\left\{\left\lceil \frac{c_P * \hat{a} * \mathrm{MTU}_P}{\sum_i c_i} \right\rceil, \mathrm{MTU}_P\right\} & (c_P > s_P \wedge p_P \geq c_P) \end{cases}.$$

Like for CMT/RP, the ceiling function here also ensures an increase of at least one byte (see also Subsubsection 8.3.1.2). $\hat{a}$ denotes the per-flow aggressiveness factor, which is defined as:

$$\hat{a} = \left(\sum_i c_i\right) * \frac{\max_i\left\{\frac{c_i/\mathrm{MTU}_i}{(\mathrm{RTT}_i)^2}\right\}}{\left(\sum_i \frac{c_i/\mathrm{MTU}_i}{\mathrm{RTT}_i}\right)^2}.$$

This formula is based on [RHW09], but has been transferred from a congestion window given in TCP MSS to a congestion window given in bytes. This has been necessary, since the congestion windows of SCTP are counted in bytes, as explained in Section 3.8.

Furthermore, the congestion window decrease behaviour has been slightly modified. In case of a retransmission (i.e. fast or timer-based) on path P, s_P and c_P are reduced as follows:

$$\begin{aligned} s_P &= \max\left\{c_P - \frac{1}{2} * c_P, \mathrm{MTU}_P\right\}, \\ c_P &= \begin{cases} s_P & \text{(Fast Retransmission)} \\ \mathrm{MTU}_P & \text{(Timer-Based Retransmission)} \end{cases}. \end{aligned}$$

The difference to original SCTP congestion control – as defined in Section 3.8 – is very small. Here, only the lower limit for s_P – and therefore also for c_P – is reduced from $4 * \mathrm{MTU}_P$ to $1 * \mathrm{MTU}_P$. The reason for this modification will be explained in the following Section 8.4.

8.4 The Challenge of Chunk-Based Segmentation

As introduced in Section 3.6, SCTP segments messages into DATA chunks according to the smallest path MTU of the association. Once a DATA chunk has been created, it remains atomic and the assigned TSN cannot be changed. This means that the DATA chunks must be sent as a whole; a change of the segmentation – e.g. into smaller pieces – is not possible. That is, DATA chunks leading to MTU-sized packets imply the transmission of such packets. Therefore, SCTP uses a lower bound of MTU_P for the congestion window c_P of path P (as defined by RFC 4960; see [Ste07, Section 7]). On the other hand, the TCP protocol (see Subsection 2.13.3) – and therefore MPTCP as well (see Section 4.3) – segments in units of bytes. That is, these protocols can flexibly adapt the payload of a packet to a size as small as one byte. Clearly, from the perspective of overhead (see Subsubsection 2.4.1.2), it is obviously useful to send MTU-sized packets to maximise the transport efficiency.

The challenge arising from the segmentation into DATA chunks is the fairness when transporting MTU-sized packets over a bottleneck shared by multiple paths. For example, let two paths share a common bottleneck with MTU_1=1,500 bytes and MTU_2=1,500 bytes, RTT_1 and RTT_2 the RTTs as well as congestion windows c_1=MTU_1 and c_2=MTU_2. That is, the congestion window sizes are at their minimum. Having a saturated sender, the used bandwidths of each path will not fall below $\frac{\mathrm{MTU}_1}{\mathrm{RTT}_1}$ and $\frac{\mathrm{MTU}_2}{\mathrm{RTT}_2}$, respectively (see also Subsubsection 2.9.2.3) – regardless of any further loss events. That is, whenever the congestion window of a path is at its minimum, at least a stop and wait with MTU-sized packets on this path remains possible (see also Subsection 2.9.1). The higher the number of paths, the higher the minimum used bandwidth – regardless of the congestion caused. When using bytes as segments, as for TCP/MPTCP, the sender could simply generate smaller packets – although, of course, this would reduce the efficiency of the transport.

In order to cope with the problem of the minimum congestion window necessary for CMT-SCTP, a simple approach – denoted as *RP Path Blocking* – has been developed as part of this thesis. On each path P, the RP-based congestion control is extended as follows:

- The RP-based congestion control mechanisms (CMT/RPv1, CMT/RPv2, MPTCP-like) are applied as introduced in Section 8.3.

- However, each time the congestion window c_P would be reduced to a value less than MTU_P (which is prevented by the minimum function), the path P enters the "RP Blocked" state for one RTO on path P. That is, with the default setting of RTO.Min=1 s (see Section 3.8), this state will persist for at least 1 s.

- As long as the path P is in the "RP Blocked" state, it will not be used for transmitting new DATA chunks. It can only be used for timer-based retransmissions as well as for control chunks (particularly, HEARTBEAT and HEARTBEAT_ACK chunks for the path monitoring; see also Subsection 3.4.3). These exceptions are useful to retain path redundancy in case of path failures.

8.5 Scenario Setup

In order to show the performance effects of RP-based congestion control for CMT-SCTP, the simulation model introduced in Chapter 5 has been applied again. The two simulation scenarios used in this chapter are depicted in Figure 8.1:

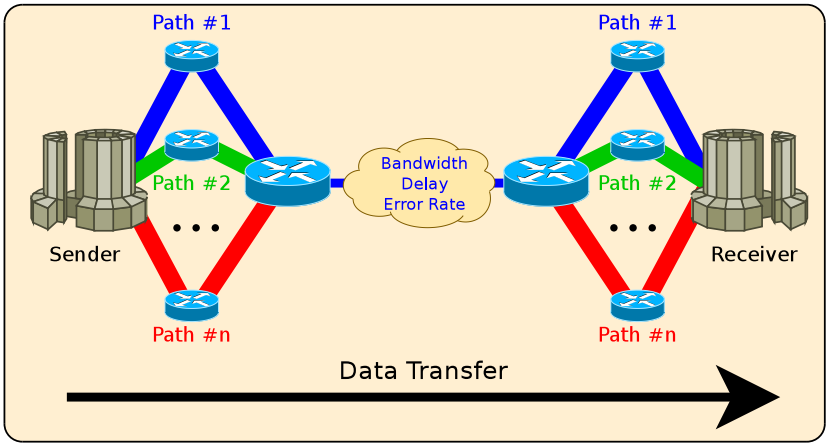
- Subfigure 8.1(a) presents the shared bottleneck scenario, with all n paths sharing the same bottleneck link.

- Subfigure 8.1(b) shows the disjoint path scenario, where all n paths are independent.

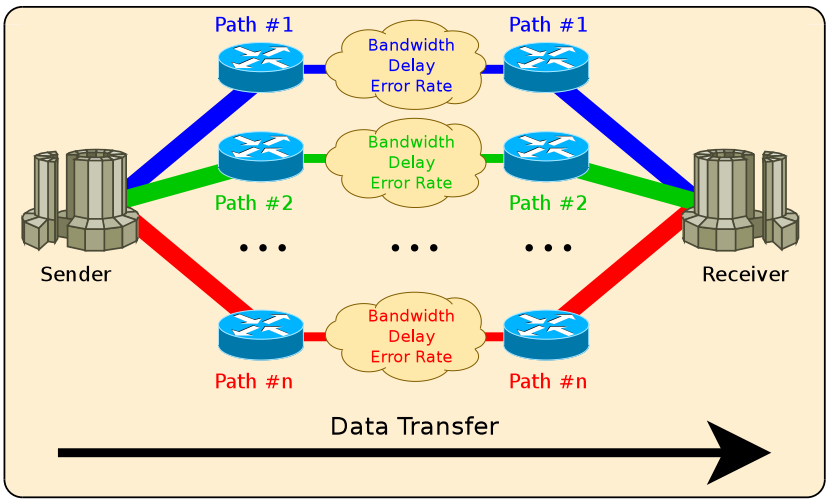
Clearly, CMT-SCTP associations use all n paths. In order to realise concurrency, non-CMT SCTP associations denoted as reference flows only use Path #n (i.e. the highest-numbered path). That is, the concurrency between CMT-SCTP and non-CMT-SCTP flows occurs on this path only.

Unless otherwise specified, the SCTP parameters introduced in Section 7.2 for the dissimilar paths performance evaluations of Chapter 7 have been used with the following adaptations:

- The delay of each path has been 10 ms. This is a reasonable setting for a WAN setup without buffer bloat.

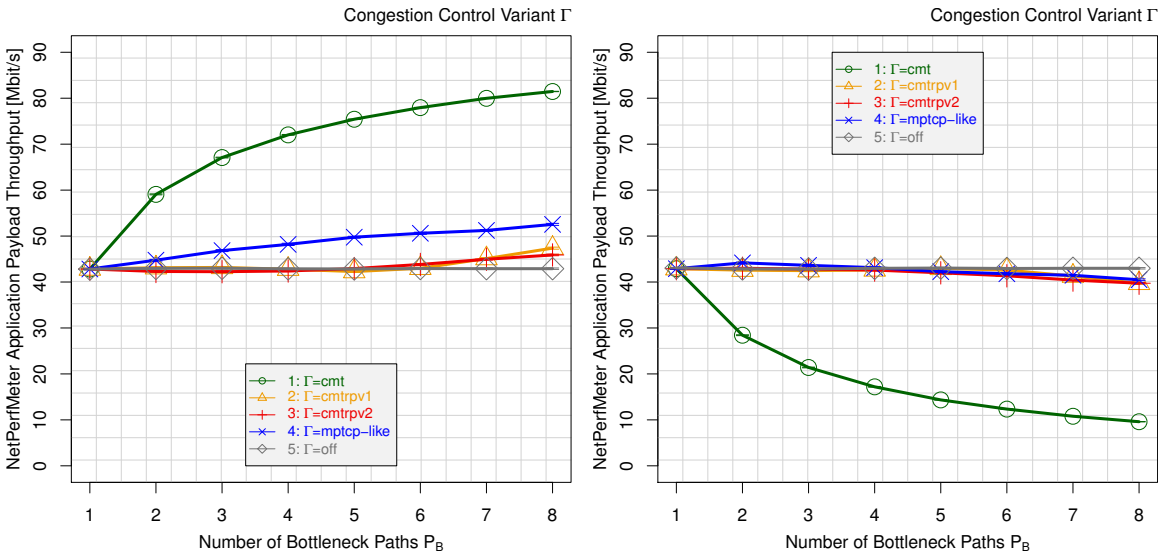


(a) Shared Bottleneck



(b) All Paths Disjoint

Figure 8.1: The Scenario Setups for the Fairness Evaluations



(a) CMT-SCTP Flow

(b) Non-CMT Reference Flow

Figure 8.2: Concurrency between CMT and Non-CMT Flow on a Shared Bottleneck

- There has been one CMT-SCTP flow and one non-CMT reference flow. Both senders, i.e. the CMT-SCTP as well as the non-CMT-SCTP flow, have been saturated. Messages of 1,452 bytes have been sent with unordered delivery at an MTU of 1,500 bytes, leading to MTU-sized packets.
- The send buffer and receive buffer sizes have been set to 5,000,000 bytes. They have been large enough to cope with the scenarios used in this chapter (as described in detail in Section 7.4).
- For the RP-based congestion controls, RP path blocking (as introduced in Section 8.4) has been applied.
- The runtime of each simulation run has been 300 s.

8.6 Handling Shared Bottlenecks

Clearly, the motivation of RP has been the fairness – according to the three RP goals defined in Section 8.2 – on shared bottlenecks.

8.6.1 Varying the Number of Bottleneck Paths

In order to demonstrate the performance impact of a shared bottleneck, the initial simulation has varied the number of bottleneck paths P_B in the scenario depicted in Subfigure 8.1(a). The bandwidth of the bottleneck has been 100 Mbit/s. Figure 8.2 presents the achieved application payload throughput results of the CMT-SCTP flow (in Subfigure 8.2(a)) and the non-CMT reference flow (in Subfigure 8.2(b)) for the different congestion control variants.

For comparison, curve 5 (drawn in gray colour) shows the performance for CMT turned off, i.e. using standard SCTP as defined in RFC 4960 (see [Ste07]) for both flows. Obviously, the achieved

throughput of both flows is equal and independent of the number of bottleneck paths P_B. Note, that the per-flow throughput of 43 Mbit/s is a result of the short RED queue (as explained in Subsection 2.4.2) using MinTh=30, i.e. it behaves like a FIFO queue for up to 30 packets. At 100 Mbit/s, 30 packets at an MTU of 1,500 bytes result in a delay of only

$$\frac{30 \text{ packets}}{\frac{(100{*}10^6 \text{ bit/s})/(8 \text{ bit/B})}{1{,}500 \text{ B/packet}}} = 3.6 \text{ ms}.$$

That is, this configuration ensures a low-delay communication by avoiding buffer bloat (see Subsection 7.6.3). However, the effect on the congestion windows of the flows is that they remain small. Losses, caused by the concurrency, lead to short phases of underutilisation. Therefore, the resulting throughput is slightly less than the theoretically achievable value of $\frac{95 \text{ Mbit/s}}{2}$=47.5 Mbit/s. The impact of a significantly longer queue will be examined in Subsection 8.6.3.

When turning on CMT-SCTP, all four CMT congestion control variants (curves 1 to 4) obviously show no difference for having only one path (i.e. P_B=1). Furthermore, the results for plain CMT-SCTP congestion control (i.e. curve 1, drawn in dark green colour) indicates the expected unfairness issue for a rising number of bottleneck paths. That is, the CMT-SCTP throughput at P_B=8 is more than 80 Mbit/s (curve 1 in Subfigure 8.2(a)), while the non-CMT flow not even reaches 10 Mbit/s (curve 1 in Subfigure 8.2(b)).

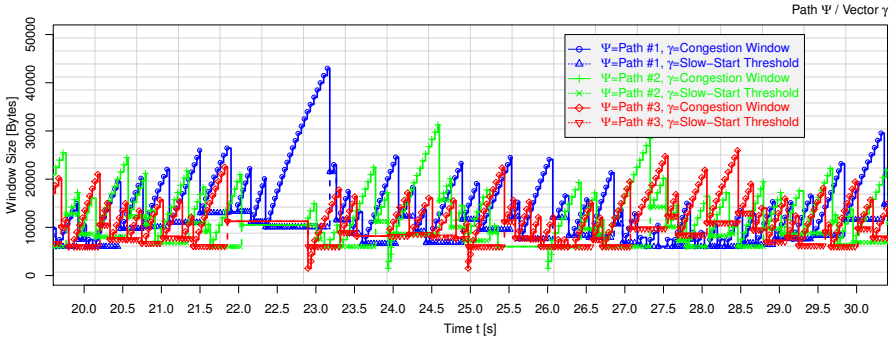
By using CMT/RP – either in version 1 (curve 2, drawn in orange colour) or version 2 (curve 3, drawn in red colour) – the fairness issue is solved for a reasonably small number of bottleneck paths P_B. For $P_B \leq 4$, no significant throughput difference between the CMT and the non-CMT flow is observable. Higher settings of P_B show a slowly rising difference, but – in comparison to plain CMT-SCTP (curve 1) – it remains small. At P_B=8, the non-CMT reference flow still achieves a throughput of 40 Mbit/s instead of less than 10 Mbit/s. Note, that there is no significant difference between CMT/RPv1 and CMT/RPv2 in this case: all paths share the same bottleneck and have comparable slow-start thresholds (see also Subsubsection 8.3.1.2).

The most interesting result is for MPTCP-like congestion control (curve 4, drawn in blue colour). For the non-CMT reference flow (see Subfigure 8.2(b)), the performance is similar to CMT/RPv1 and CMT/RPv2 (curves 2 and 3). However, the throughput of the CMT flow (see curve 1 in Subfigure 8.2(a)) is improved by the increasing number of bottleneck paths P_B. The reason here is that the MPTCP-like congestion control makes use of the underutilisation on the shared link. That is, it takes the remaining bandwidth which is not utilised by the non-CMT reference flow. In order to further explain this interesting effect, it is necessary to have a closer look at the congestion control behaviour of the four variants.

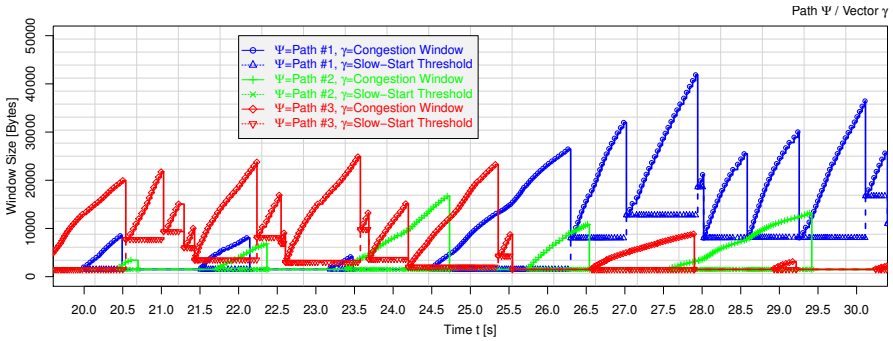
8.6.2 Congestion Control Behaviour on Bottleneck Paths

In order to further explain the congestion control effects observed for the simulations in Subsection 8.6.1, Figure 8.3 presents the values of congestion window c_i and slow-start threshold s_i of Path #i for three bottleneck paths (i.e. P_B=3) during the time interval from t=20 s to t=30 s.

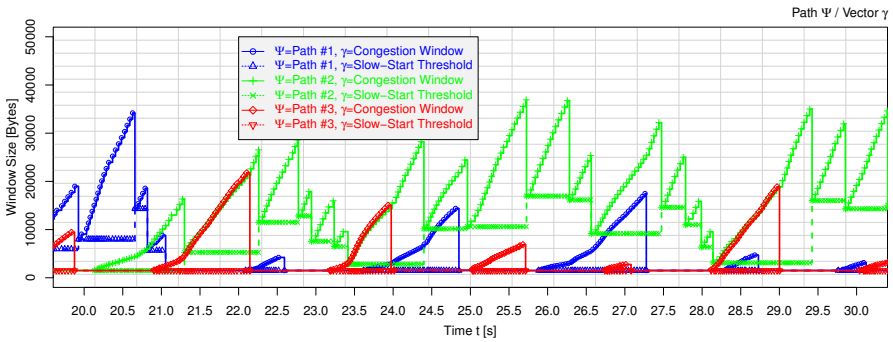
Clearly, for plain CMT-SCTP congestion control as presented by Subfigure 8.3(a), each path behaves like an individual non-CMT flow. An interesting effect of the shared bottleneck can be observed at around t=23.8 s: timer-based retransmissions on Path #2 (in green colour) and Path #3 (in red colour) after RTOs of 1 s. This is a result of the concurrency on the bottleneck. Since these paths have reduced their bandwidth before, Path #1 (in blue colour) is able to achieve a significantly larger congestion window here. The synchronised occurrence of such congestion incidents (here: for Path #2



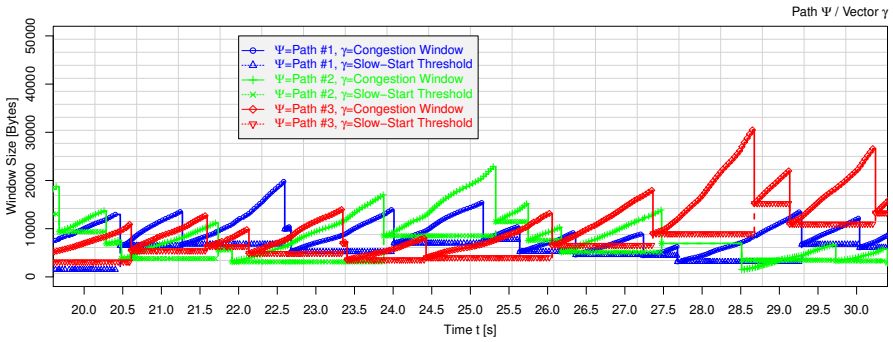
(a) Plain CMT-SCTP Congestion Control



(b) CMT/RPv1 Congestion Control



(c) CMT/RPv2 Congestion Control



(d) MPTCP-Like Congestion Control

Figure 8.3: The Impact of the Congestion Control Variant for Three Shared Paths

and Path #3) on shared bottlenecks is also the motivation of statistics-based detection approaches, as introduced by [YWY08b, YWY08a].

The behaviour of CMT/RPv1 (shown in Subfigure 8.3(b)) and CMT/RPv2 (shown in Subfigure 8.3(c)) is very similar, due to the comparable slow-start thresholds of the paths. The effect caused by CMT/RP – in either version – is that frequently one of the paths achieves a higher congestion window. Then, the slow-start threshold ratios $\hat{s}$ (CMT/RPv1; see Subsubsection 8.3.1.1) or increase factors $\hat{i}$ (CMT/RPv2; see Subsubsection 8.3.1.2) of the other paths become small, leading to only a slow growth rate of the corresponding congestion windows. When the congestion window of the currently powerful path drops, this effect may change to another path. Particularly, for CMT/RP, a congestion window c_P for a path P with MTU_P occurs relatively frequently, including triggering the RP path blocking mechanism (see Section 8.4). The RP path blocking is visible by a constant congestion window of $c_P = \mathrm{MTU}_P$ for one path RTO (in this case: 1 s, due to the lower limit of RTO.Min=1 s).

For MPTCP-like congestion control (shown in Subfigure 8.3(d)), the behaviour is significantly different from CMT/RP. This congestion control approach tries to balance the congestion windows, leading to a more similar behaviour among the paths. In consequence, since the congestion window c_P of a path P only rarely falls down to MTU_P, RP path blocking also needs to be triggered very infrequently. Therefore, a drop of one of the congestion windows only leads to a slight reduction of the throughput, which explains the ability of MPTCP-like congestion control to make use of the otherwise underutilised bandwidth, as observed in Subsection 8.6.1.

8.6.3 Using a Long Queue before the Bottleneck

The simulation presented in Subsection 8.6.1 has used a short RED queue at the bottleneck, which has led to a low delay but also to a slight underutilisation of the bottleneck link. In order to show the effect of a long queue, the RED queue parameters (as explained in Subsection 2.4.2) have been adapted:

- The bottleneck link delay has been 10 ms, i.e. the RTT has been 20 ms. This time span can be covered by setting MinTh=168:

$$\frac{168 \text{ packets}}{\frac{(100{*}10^6 \text{ bit/s})}{(8 \text{ bit/B})}} = 20.16 \text{ ms}.$$

- According to the recommendations on RED queue configuration by [Flo97], MaxTh has been set to 3*MinTh, i.e. MaxTh=504. The queue filled with 504 MTU-sized packets would therefore introduce an additional delay of 60.48 ms.

The achieved application payload throughput for varying the number of bottleneck paths P_B is presented in Figure 8.4. To enhance readability, the curves for the plain CMT-SCTP congestion control case are omitted and the Y-axis is zoomed to the interesting range from 30 Mbit/s to 60 Mbit/s. The performance of the CMT-SCTP flow is shown by solid lines (F=1); the non-CMT reference flow throughput by dashed lines (F=2). These results, in an adapted form, have also been published in [DBAR11b].

When CMT-SCTP is completely turned off, as shown for comparison by curves 7 and 8, the changed queue configuration now ensures that the bandwidth of the bottleneck link can be fully utilised. Both flows, applying standard SCTP as defined in RFC 4960 (see [Ste07]) reach the expected throughput of about 47.5 Mbit/s. However, due to the large buffers, also the delay is significantly increased. That is, the throughput is maximised by a slight buffer bloat (see also [Get11a, Get11b]).

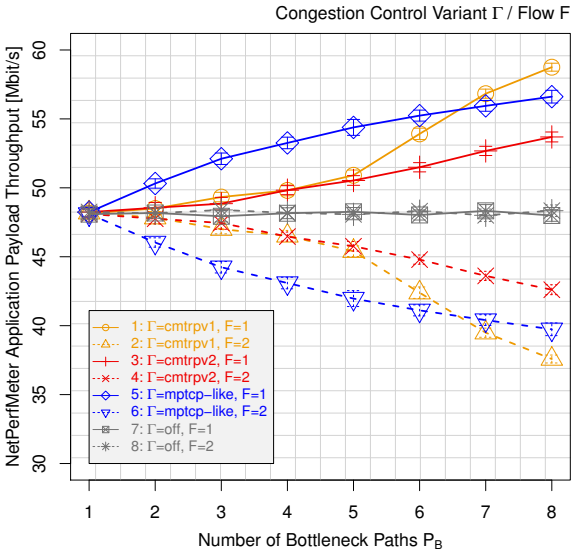


Figure 8.4: Concurrency on a Shared Bottleneck with a Long Queue

While the delay increase here is still relatively small, this effect occurs to a much greater extent in DSL setups as described in Subsection 7.6.3. Curves 7 and 8 show an almost perfect throughput share between both flows. This is the baseline performance for the RP-based congestion control variants.

The application payload throughput results for CMT/RPv1 (curves 1 and 2) show an increasing divergence for a rising number of bottleneck paths. Particularly, for P_B>5, this divergence significantly increases. The reason here is that paths become temporarily dissimilar, particularly also due to the application of RP path blocking (see Section 8.4). This makes the slow-start threshold ratios (see Subsubsection 8.3.1.1) of the paths less comparable, leading to an inappropriate increase behaviour on the paths. In consequence, the throughput of the non-CMT reference flow is significantly reduced, e.g. from the expected 47.5 Mbit/s to 38 Mbit/s at P_B=8.

Obviously, CMT/RPv2 (curves 3 and 4) significantly improves the performance in these situations. Here, the divergence between the flows increases linearly with the number of bottleneck paths P_B. Using the bandwidth of the paths to adapt the AIMD behaviour – as described in Subsubsection 8.3.1.2 – lets the congestion control behaviour adapt to the temporary dissimilarity of the paths. At P_B=8, the throughput of the reference flow only reduces from the expected 47.5 Mbit/s to about 43 Mbit/s.

Similar to CMT/RPv2, also the MPTCP-like congestion control (curves 5 and 6) shows a relatively linear increase of the divergence between the throughputs of both flows. However, the difference is stronger than for CMT/RPv2. That is, the throughput of the non-CMT reference flow is reduced to about 40 Mbit/s at P_B=8. The reason for this behaviour is the higher aggressiveness of MPTCP-like congestion control. Since it tries to balance the size of the congestion windows, as explained in Subsection 8.6.2, the case of reaching the minimum of $c_P = \mathrm{MTU}_P$ for a path P is rare. This also means that RP path blocking is rarely triggered (see Section 8.4).

Despite the slight throughput divergence for an increasing number of bottleneck paths, it is important to note that CMT/RPv2 and MPTCP-like congestion control still reach a significant improvement on fairness for CMT-SCTP transport. In comparison, the throughput of the non-CMT reference flow when applying plain CMT-SCTP congestion control would be less than 10 Mbit/s for P_B=8 bottleneck paths, which is less than one quarter of the performance achieved with CMT/RPv2 or MPTCP-like

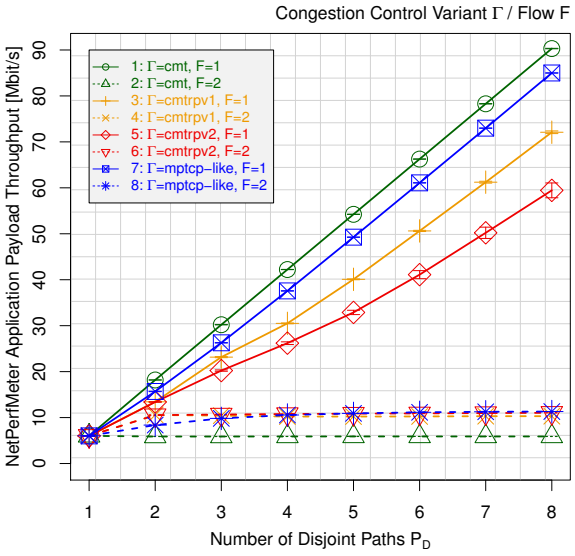


Figure 8.5: Concurrency between CMT and Non-CMT Flow on Disjoint, Similar Paths

congestion control. That is, the RP-based congestion controls fulfil the first two goals set in Section 8.2 – which are utilisation and fairness – reasonably well. For the third goal, i.e. congestion balancing, it is necessary to examine disjoint path setups.

8.7 Handling Disjoint Paths

In order to analyse the performance on disjoint paths, it is useful to first examine a similar path setup which uses the setup illustrated in Subfigure 8.1(b). The application payload throughput performance results for varying the number of disjoint paths P_D, each path having a bandwidth of 12.5 Mbit/s, is presented in Figure 8.5. The curves for the CMT-SCTP flow (F=1) are drawn as solid lines; the curves for the non-CMT reference flow (F=2) as dashed lines.

Obviously, the results for plain CMT-SCTP congestion control (curves 1 and 2) are not very surprising: the CMT-SCTP flow can exclusively utilise $P_D - 1$ paths plus half of the bandwidth on Path #P_D, i.e. the path shared with the reference flow. In result, the reference flow achieves the expected application payload throughput of about 6 Mbit/s.

CMT/RPv1 as well as CMT/RPv2 ensure that the non-CMT reference flow achieves a significantly higher throughput (curves 4 and 6) when the scenario becomes multi-homed (i.e. $P_D \geq 2$). The throughput rises to about 11 Mbit/s at P_D=2 and remains almost constant for a higher number of paths. That is, since the CMT-SCTP flow can utilise $P_D - 1$ paths exclusively, it gives most of the bandwidth (i.e. 12.5 Mbit/s) on the path shared with the non-CMT flow to the non-CMT flow. This fulfils the third goal, i.e. congestion balancing, defined for resource pooling in Section 8.2. However, the bandwidth on the exclusively used paths is not fully utilised (curves 3 and 5). At P_D=8, CMT/RPv1 only achieves about 72 Mbit/s, while CMT/RPv2 just gets about 60 Mbit/s. The reason for the difference between the two variants is that the path bandwidth, on which the CMT/RPv2 AIMD behaviour is based, is significantly more varying in this setup. On the other hand, the slow-start threshold ratio, which is used for CMT/RPv1, remains very stable on the exclusively used paths.

MPTCP-like congestion control, on the other hand, is able to fully utilise the bandwidth on the

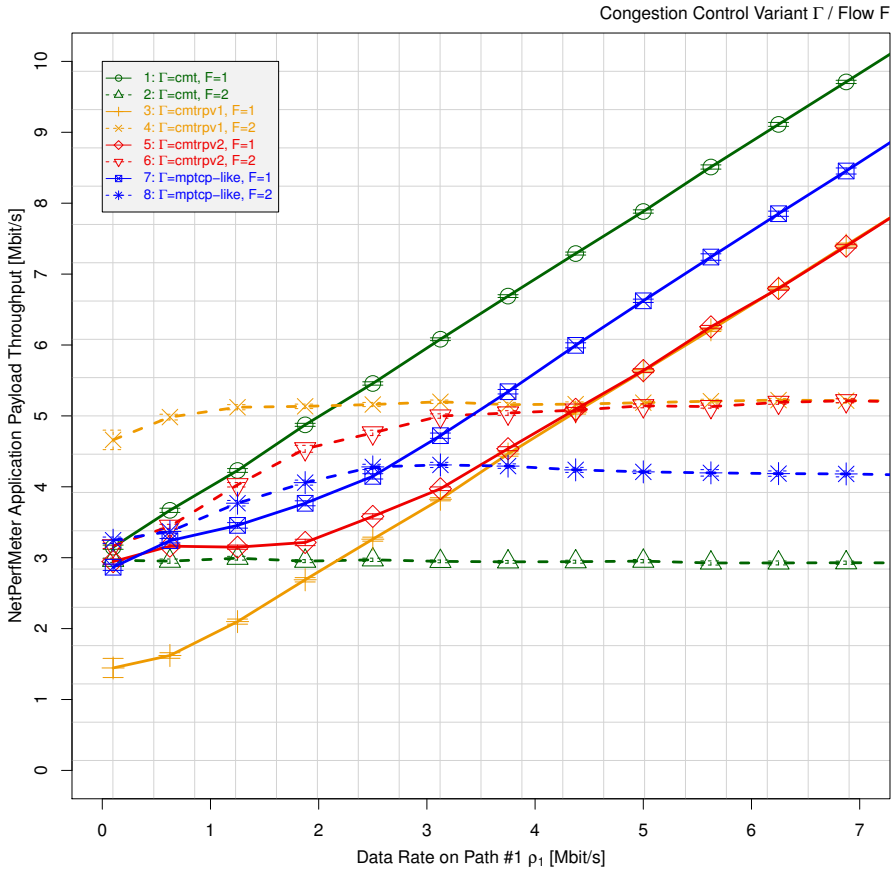


Figure 8.6: Bandwidth Variation on Path #1 being Exclusively Used by the CMT-SCTP Flow

exclusively used paths (curve 7), while it also gives about 11 Mbit/s to the non-CMT reference flow (curve 8) for higher settings of P_D (here: $P_D \geq 4$). For a smaller number of paths, it is slightly more aggressive. However, in all cases, the performance of the non-CMT flow is significantly improved in comparison to using plain CMT-SCTP congestion control (curve 2). In order to further explain the different behaviours of the congestion control variants, it is useful to have a look at disjoint paths that are dissimilar.

8.8 Dissimilar Paths

In order to demonstrate the impact of dissimilar paths, the disjoint paths scenario illustrated in Subfigure 8.1(b) has been used with two paths. The following results – in a slightly adapted form – have also been published in [DBAR11b].

8.8.1 Bandwidth Variation on the Exclusively Used Path

In the first scenario, the bandwidth ρ_1 of Path #1 – which is exclusively used by the CMT-SCTP flow – has been varied. The bandwidth of Path #2, which is shared with the non-CMT reference flow, has been fixed at ρ_2=6.25 Mbit/s. Figure 8.6 presents the resulting application payload throughput

performance results; the curves for the CMT-SCTP flow (F=1) are drawn as solid lines, the curves for the non-CMT reference flow (F=2) as dashed lines.

The results for plain CMT-SCTP congestion control are as expected. The CMT-SCTP flow (curve 1) can utilise half of the bandwidth of Path #2 as well as the complete bandwidth of Path #1. Therefore, the non-CMT reference flow (curve 2) takes the other half of the bandwidth on Path #2, achieving the expected application payload throughput of about 3 Mbit/s. Clearly, this is the baseline performance level.

CMT/RPv1 fails to meet this baseline performance for the CMT-SCTP flow (curve 3). For a Path #1 bandwidth of $\rho_1 \leq 2$ Mbit/s, its throughput is smaller than 3 Mbit/s. On the other hand, the non-CMT reference flow can always achieve a throughput of about 5 Mbit/s (curve 4). This obviously violates the utilisation goal (see Section 8.2), i.e. the non-CMT flow should at least get a bandwidth share which is as large as the share of a non-CMT flow on the best path (which is Path #2 in this case). The reason for the performance problem of CMT/RPv1 is the dissimilarity of the paths, which is particularly problematic for small settings of the Path #1 bandwidth ρ_1. Path #1 is exclusively used by the CMT-SCTP flow, i.e. its congestion window can grow large – particularly also due to the RED queue with MinTh=30, which causes a slight buffer bloat for the small bandwidths ρ_1 (see Subsection 7.6.3). On the shared Path #2, losses due to the concurrency are frequent. Since the slow-start threshold on this path is significantly smaller therefore, chances are good that the congestion window reduction (see Subsubsection 8.3.1.1) also triggers the RP path blocking mechanism (see Section 8.4), preventing the usage of this path for one RTO on this path (i.e. at least RTO.Min=1 s). Obviously, CMT/RPv1 is not useful for dissimilar paths.

CMT/RPv2 can cope with the dissimilar paths as expected. Here, the performance of the CMT-SCTP flow (curve 5) is always better than the baseline of 3 Mbit/s. The bandwidth of the non-CMT reference flow (curve 6) increases with a rising Path #2 bandwidth ρ_2 until about 5 Mbit/s. That is, when there is sufficient bandwidth on the exclusively used Path #1, the non-CMT reference flow can use most of the Path #2 bandwidth. This fulfils the goal of congestion balancing, as defined in Section 8.2. Note, that particularly from $\rho_1 \geq 1$ Mbit/s to $\rho_1 \leq 4$ Mbit/s, the non-CMT reference flow achieves a significantly higher bandwidth than the CMT-SCTP flow. This interesting effect of congestion balancing does not violate the goals, since both flows achieve a better throughput than for using Path #1 alone. It will be discussed in more detail in Subsection 8.8.2.

The same effect, although weaker, is also observable for the MPTCP-like congestion control results (curves 7 and 8). As already observed before, the MPTCP-like variant is more aggressive, occupying a larger share of the Path #2 bandwidth for the CMT-SCTP flow. Therefore, at a Path #1 bandwidth of ρ_1=7 Mbit/s, the achieved throughput of the non-CMT reference flow is about 4.25 Mbit/s (curve 8), in contrast to the about 5 Mbit/s achieved by CMT/RPv2 (curve 6). Clearly, also MPTCP-like congestion control fulfils the goals defined in Section 8.2.

While this simulation has varied the bandwidth of the exclusively used path, the effects of varying the bandwidth of the other path is also interesting.

8.8.2 Bandwidth Variation on the Shared Path

In order to examine the effects of varying the bandwidth of the path shared between the CMT-SCTP flow and the non-CMT reference flow, the bandwidth of Path #1 has been fixed at ρ_1=6.25 Mbit/s, while the bandwidth ρ_2 of Path #2 has been varied from 0.25 Mbit/s to 30.0 Mbit/s. Figure 8.7 presents the resulting application payload throughput results for the CMT-SCTP flow (F=1; solid lines) as well as for the non-CMT reference flow (F=2; dashed lines).

The performance of plain CMT-SCTP congestion control is as expected. The non-CMT reference

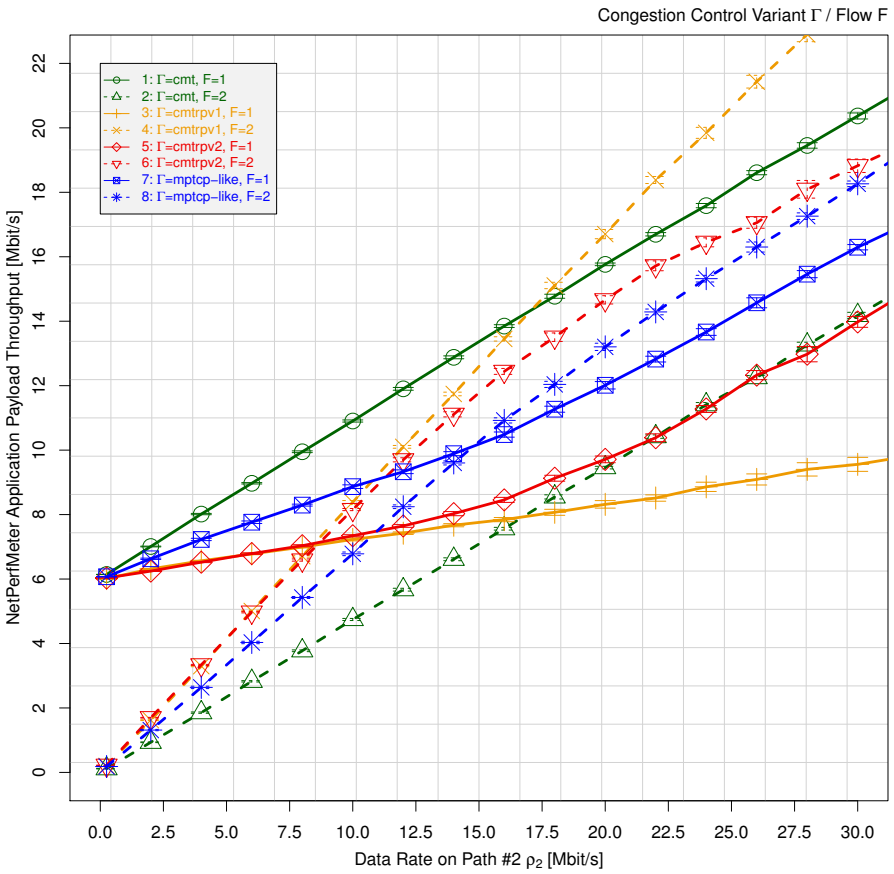


Figure 8.7: Bandwidth Variation on Path #2 being Shared with the Non-CMT Reference Flow

flow (curve 2) and the CMT-SCTP flow (curve 1) equally share the bandwidth of Path #2, while the bandwidth of Path #1 is exclusively used by the CMT-SCTP flow. Therefore, the curves of both flows are parallel. These two parallels define the baseline performance levels.

As already observed and explained in Subsection 8.8.1, CMT/RPv1 cannot cope appropriately with dissimilar paths. Therefore, for about $\rho_2 \geq 17.5$ Mbit/s, the application payload throughput for the CMT-SCTP flow falls below the baseline performance, while the non-CMT reference flow exceeds it.

CMT/RPv2 can handle the path dissimilarity. However, the effect of congestion balancing as already observed in Subsection 8.8.1 results in the CMT-SCTP flow performance (curve 3) converging towards the lower baseline performance level (i.e. curve 2) for a higher dissimilarity of the paths (here: for $\rho_2 \geq 20$ Mbit/s). On the other hand, the performance of the non-CMT reference flow is significantly higher but cannot fully reach the upper baseline performance level (i.e. curve 1). Here, the high bandwidth in combination with the relatively short RED queue (i.e. MinTh=30) results in the effects for short queues that have been observed and explained in detail in Subsection 8.6.1. That is, some bandwidth remains underutilised with this queue size. Nevertheless, CMT/RPv2 still fulfils the three goals set in Section 8.2: the performance for CMT-SCTP is not worse than for a non-CMT flow on the best path (utilisation), the non-CMT reference flow can even increase its bandwidth (fairness) and load is moved away from the shared path (congestion balancing).

The MPTCP-like congestion control performs slightly better in this scenario, by providing a higher bandwidth for the CMT-SCTP flow (curve 7). Furthermore, for higher settings of the Path #2 bandwidth ρ_2 (here: $\rho_2 \geq 25$ Mbit/s), it also achieves a throughput for the non-CMT reference flow (curve 8) which is almost as high as for using CMT/RPv2 (curve 6). That is, as observed and explained in Subsection 8.6.1, MPTCP-like congestion control can better utilise the bandwidth despite of the short RED queues. Similar to CMT/RPv2, also the MPTCP-like congestion control variant fulfils the goals set in Section 8.2.

Similar to CMT/RPv2, the achieved throughput of the non-CMT reference flow (curve 8) is better than the throughput of the CMT-SCTP flow (curve 7) for $\rho_2 \geq 16$ Mbit/s. This effect of congestion balancing does not violate the goals, since both flows perform at least as good as single-homed flows on the best path. However, for the user of CMT-SCTP, the question may arise why to have an increased effort (maintaining multiple paths, larger send/receive buffers, etc.) to slightly improve the own throughput, while the non-CMT reference flow user gets a significant performance improvement for free. On the other hand, from a global perspective, e.g. by the administrator of the network, this observed behaviour may be quite useful: the bandwidth among the flows is more evenly distributed. Clearly, these considerations point out to a need for further research on fairness with special respect to multipath transfer and load balancing. This will be a very interesting topic for the ongoing and future work.

8.8.3 Congestion Control Behaviour on Dissimilar Disjoint Paths

Finally, in order to further explain the differences between the congestion control variants, it is again useful to have a closer look at the congestion control behaviour. Figure 8.8 presents the plots of congestion windows and slow-start thresholds from time t=20 s to t=30 s for the CMT-SCTP flow in the simulation scenario described in Subsection 8.8.2.

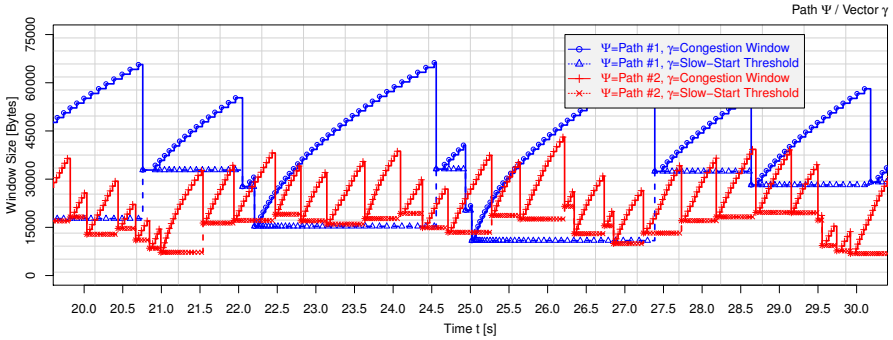
- The bandwidth of Path #1 has been 6.25 Mbit/s; the bandwidth of Path #2 has been 12.5 Mbit/s.
- Concurrency with the non-CMT reference flow has been on Path #2.

The behaviour of plain CMT-SCTP congestion control, as shown in Subfigure 8.8(a), is as expected. Both disjoint paths are handled independently. Note, that the heights of the congestion window sawteeth on the two paths are different, due to the path dissimilarity.

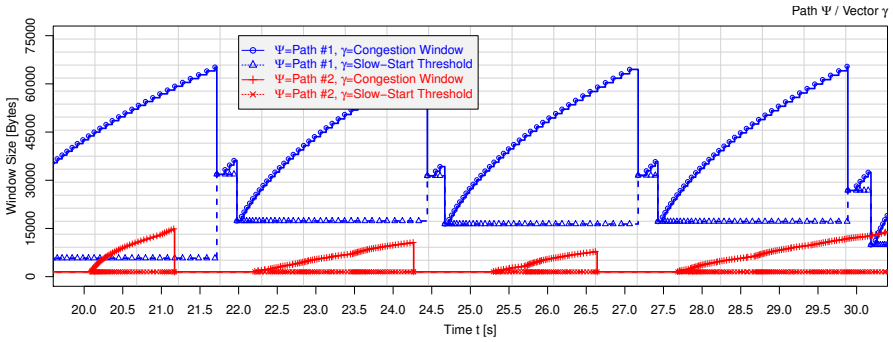
The plot for CMT/RPv1 in Subfigure 8.8(b) illustrates the lack of this congestion control variant to cope with dissimilar paths. The congestion window of Path #2 can hardly grow, due to the minimal slow-start threshold – and therefore a bad slow-start threshold ratio (see Subsubsection 8.3.1.1) – on this path. Furthermore, after each congestion window drop on Path #2, RP path blocking is triggered, leading to a paused transmission on this path for one path RTO (here: for RTO.Min=1 s).

CMT/RPv2 – as shown in Subfigure 8.8(c) – can better utilise Path #2 than CMT/RPv1. However, in comparison to plain CMT-SCTP congestion control (see Subfigure 8.8(a)), the congestion window of Path #2 still remains quite small. Particularly, also RP path blocking can be observed. This is the reason for restrainedly claiming bandwidth on this path, leaving a larger share for the non-CMT reference flow (see also Subsection 8.8.2).

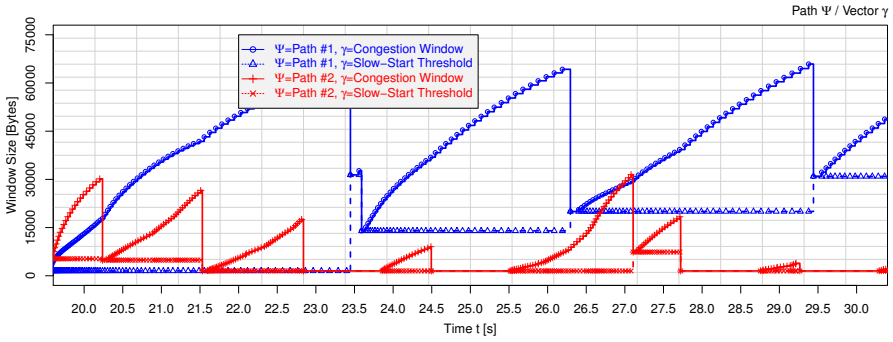
Finally, the MPTCP-like congestion control variant shows its expected behaviour, as presented in Subfigure 8.8(d). While the congestion window on Path #2 is still small in comparison to plain CMT-SCTP congestion control (see Subfigure 8.8(a)) – as it is intended – it does not drop down to the minimal value (i.e. one path MTU). Therefore, within this plot, also no occurrence of RP path blocking is observable. The higher level of the congestion window values is the reason for the higher



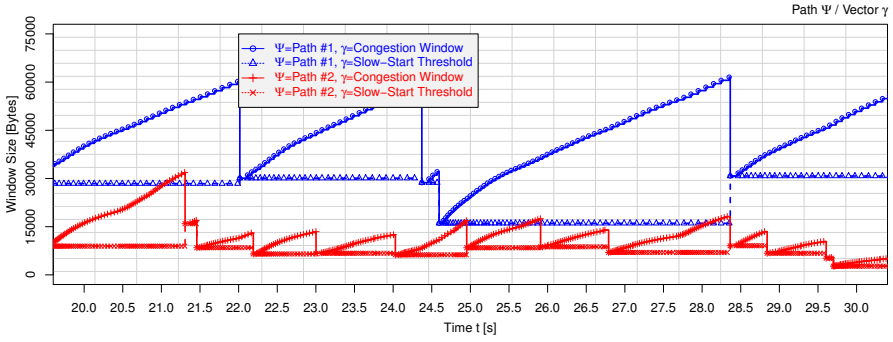
(a) Plain CMT-SCTP Congestion Control



(b) CMT/RPv1 Congestion Control



(c) CMT/RPv2 Congestion Control



(d) MPTCP-Like Congestion Control

Figure 8.8: The Impact of the Congestion Control Variant for Two Disjoint, Dissimilar Paths

throughput on Path #2, which leads to a reduced throughput for the non-CMT reference flow (see also Subsection 8.8.2).

8.9 Ongoing and Future Work

The simulative analyses of the RP-based congestion controls have shown their usefulness. Clearly, as ongoing work, it is also necessary to evaluate these variants in the testbed. While CMT/RPv1 had already been contributed to the FreeBSD release 8.2, CMT/RPv2 as well as the MPTCP-like congestion controls have now also been realised in the FreeBSD kernel SCTP implementation. They are currently evaluated in the local as well as the distributed Internet testbed environments introduced in Chapter 6. The goal for the near future is to make the usage of CMT-SCTP in combination with a RP-based congestion control variant a "safe default" setting for an SCTP implementation, i.e. to activate it by default to make use of load sharing if possible, but cause no harm, i.e. significant unfairness, if paths are shared.

Clearly, future work will further refine the RP-based congestion control variants. Particularly, this work will also include theoretical considerations on more exact and formal definitions of fairness, with special reference to multipath transfer. Such refined fairness definitions will allow a better tuning of the performance, e.g. to reward the increased management complexity of multiple paths by a higher throughput, to more equally balance throughputs regardless of the number of paths, etc..

Furthermore, a performance evaluation of CMT-SCTP versus MPTCP, with respect to fairness, has currently started. Clearly, similar to the equal congestion control behaviour of SCTP versus TCP, the same seems to be useful for their respective multipath transfer variants, too.

8.10 Summary

This chapter has first introduced the concept of resource pooling to handle the fairness issue on shared bottlenecks. Furthermore, three congestion control variants based on this concept have been introduced: CMT/RPv1 and CMT/RPv2 as well as the MPTCP-like congestion control which has been adapted from MPTCP to CMT-SCTP.

The CMT/RPv2 as well as the MPTCP-like congestion control have shown to solve the fairness issue reasonably well. Furthermore, both variants can also handle disjoint path setups with dissimilar paths. While MPTCP-like congestion control is more aggressive, which leads to a better utilisation of the path bandwidths, CMT/RPv2 achieves a slightly better fairness in certain setups. As part of future work, a more exact definition of "fairness" seems to be useful, in order to better tune the performance of congestion control for multipath transfer.

Chapter 9

Conclusion and Outlook

Finally, in the last chapter of this thesis, the achieved goals and results – on the efficient handling of dissimilar paths as well as the fairness on shared bottlenecks – are shortly summarised. Furthermore, an outlook to interesting ongoing and future work as well as open issues is provided.

9.1 Achieved Goals and Obtained Results

9.1.1 Simulation Environment and Testbed Environment

As initial steps for this thesis, the simulation environment as well as the testbed environment have been designed and developed. These tools have been necessary, in order to appropriately perform the research task. Both have been designed with reusability in mind, i.e. they will be used to process the future work on CMT-SCTP.

Therefore, the OMNeT++-based simulation models introduced in Chapter 5 will also be contributed to the INET framework as open source in the very near future. That is, the CMT-SCTP model (see Section 5.4), the NetPerfMeter protocol-independent application model (see Section 5.5) as well as the multi-homed auto-routing module (see Section 5.6) will become freely available for all interested researchers. Details on these models have also been published in [DBPR10b]. Furthermore, the SimProcTC tool-chain for the – possibly distributed – processing of simulation runs (to be introduced in detail in Appendix B) is already publicly available as open source. An overview has been published in [DZR09].

The NetPerfMeter network performance test application introduced in Section 6.3, which has been developed for the testbed environment described in Chapter 6, is also available as open source. It has furthermore been contributed to Debian and Ubuntu Linux as well as to the FreeBSD ports collection. That is, users can easily install it and use it for their purposes. Particularly, NetPerfMeter is also intended as multi-protocol performance test application within the G-Lab project. Details on this application, including the lessons learned on building up a distributed Internet testbed environment as described in Subsection 6.5.3, have been published in [DBAR11a]. Finally, some more lessons learned on the work with Dummynet to ensure certain QoS characteristics within a testbed can be found in [BDRF11].

9.1.2 Efficient Handling of Dissimilar Paths

Clearly, the major research part of this thesis has been the performance of CMT-SCTP in dissimilar path setups. Since SCTP provides the possibility to separately configure unordered delivery, ordered

delivery and multi-streaming, it has therefore been possible and useful to split up this part into three sub-parts.

9.1.2.1 Unordered Delivery

The first sub-part has been the performance of unordered delivery with CMT-SCTP. As a result of this thesis, the issues occurring when using plain CMT-SCTP as defined by [IAS06] have been categorised into four sub-problems:

1. Transmission-induced send buffer blocking (see Subsubsection 7.5.1.1),
2. GapAck-induced send buffer blocking (see Subsubsection 7.5.1.2),
3. Advertised-window-induced receive buffer blocking (see Subsubsection 7.5.2.1) and
4. Reordering-induced receive buffer blocking (see Subsubsection 7.5.2.2).

This categorisation has also been published in [ADB+11].

As key cause for the observed problems of unordered delivery, the unbalanced usage of send and receive buffers among the paths has been identified. The proposed solution denoted as buffer splitting, which has been introduced in Section 7.6, tackles the unbalanced buffer usage by a sender-side mechanism that takes care of the outstanding bytes on each path, with regard to send as well as receive buffer size. That is, it ensures that each path gets a sufficient share of the limited buffer space at the sender as well as at the receiver side. The parameter study of Section 7.7 has shown that the issues of transmission-induced send buffer blocking as well as advertised-window-induced receive buffer blocking are effectively prevented by buffer splitting. Furthermore, it has shown that the combination with non-renegable selective acknowledgement – which has originally been proposed for standard SCTP by [NEY+08] (see also Subsection 3.11.5)– is necessary to also solve the issue of GapAck-induced send buffer blocking. Finally, the mechanism of smart fast retransmission (see Subsection 7.7.3) has been developed to easily prevent spurious fast retransmissions. Buffer splitting as well as the motivation to apply non-renegable selective acknowledgements, smart fast retransmission and the parameter study have also been published in [DBRT10], with some further refinements to the mechanisms and testbed evaluation results in [ADB+11].

9.1.2.2 Ordered Delivery

While a receiver instance may directly forward the data using unordered delivery to the upper layer, the data using ordered delivery needs resequencing. Therefore, already received DATA chunks may have to wait in the receive buffer until all earlier chunks have been received. Dissimilar paths therefore cause the problem of reordering-induced receive buffer blocking, which is neither prevented by buffer splitting nor non-renegable selective acknowledgements. As countermeasure to this problem, the approach of chunk rescheduling (see Section 7.8) has been developed. It defines a preventive retransmission strategy which monitors the send and receiver buffer occupations at the sender instance. When there is a sign of a forthcoming reordering-induced receive buffer blocking, critical chunks are rescheduled for preventive retransmission, in order to avert a transmission stall. The parameter study in Section 7.9 shows a significant performance improvement at small additional overhead. Chunk rescheduling as well as the parameter study have also been published in [DBRT10].

The application of burst mitigation in the variant suggested by RFC 4960 has been identified as further performance issue for ordered delivery with CMT-SCTP. This variant, denoted as “Use It or

Lose It", reduces the congestion window of a path when it seems to be not appropriately utilised. Due to reordered acknowledgements, these reductions occasionally occur for dissimilar path delays, which reduce the throughput. The combination of two mechanisms has been useful:

- Smart SACK path selection (see Subsubsection 7.9.4.2) reduces the reordering of acknowledgements by the appropriate selection of a path to send an acknowledgement on.
- Applying the burst mitigation variant "Max Burst", which has been introduced by [AB05] for single-homed TCP originally, realises burst mitigation by packet counters. Therefore, the congestion window remains unaffected.

9.1.2.3 Multi-Streaming

When using multi-streaming with CMT-SCTP, the application performance can be improved by the newly introduced mechanism of predefined stream mapping (see Subsection 7.10.1). This mechanism allows a fixed mapping of streams to paths, which reduces the reordering within the streams. This leads to a reduction of buffer size requirements and message delay. Particularly, this mechanism can be applied for tunnelling applications, where different application communications are multiplexed over a single CMT-SCTP association. In this case, the different properties of dissimilar paths can be better utilised to enhance the user experience. For example, interactive streams may use a low-bandwidth but low-delay path, while non-interactive bulk data transfer streams use a high-delay but high-bandwidth path.

Furthermore, in order to take advantage of predefined stream mapping, the mechanism of decoupled streams has been developed (see Subsection 7.10.2). It is necessary to allow an application to appropriately fill the send buffer with data for each stream. The evaluation of predefined stream mapping with decoupled streams has shown the usefulness of these features. They have also been published in [DSTR10].

9.1.3 Fairness on Shared Bottlenecks

The second part of this thesis has examined the fairness of CMT-SCTP transport over shared bottlenecks. The key idea here has been to apply the concept of resource pooling introduced by [WHB08], which means to couple the congestion controls among the paths to ensure a notion of fairness which is defined together with this concept.

As a preliminary work in the context of this thesis, two new congestion control variants have been developed:

- CMT/RPv1 (based on the slow-start threshold ratios of the paths; see Subsubsection 8.3.1.1) and
- CMT/RPv2 (based on the bandwidths of the paths; see Subsubsection 8.3.1.2).

Furthermore, the approach of the congestion control proposed by [RHW09] for MPTCP, which is based on an adaptive aggressiveness, has been adapted to CMT-SCTP (see Subsection 8.3.2) and denoted as MPTCP-like congestion control. SCTP congestion control differs from TCP by two important properties:

- SCTP segments to complete chunks while TCP segments to bytes. Therefore, SCTP cannot reduce the packet size for bandwidth reduction purposes.

- The congestion control of SCTP is based on bytes, while some TCP implementations – and many congestion control approaches in literature – are based on maximum segment sizes.

The analyses in a shared bottleneck scenario have shown that CMT/RPv2 as well as MPTCP-like congestion control are useful approaches to handle the fairness issue. The difference between the throughput of a CMT-SCTP flow and a concurrent non-CMT reference flow is reasonably small. Here, MPTCP-like congestion control shows a slightly larger difference. However, on the other hand, it is better able to utilise the available bandwidth. Also, both congestion control approaches can handle disjoint and dissimilar paths. Due to its higher aggressiveness, the MPTCP-like congestion control variant is better able to utilise the available bandwidth here. Parts of these results have also been published in [DBAR11b, DBPR10a].

9.1.4 Standardisation Contributions

Besides the research part, it has also been intended to bring the developed ideas from research to application, by contributing to the ongoing IETF standardisation process for the SCTP protocol. As result of this thesis project, three Internet Drafts have been submitted as work in progress (see also Subsubsection 2.12.3.2):

- [DBA12] is an individual submission draft that introduces an API extension to configure the usage of CMT-SCTP as well as the congestion control variant in a platform-independent manner.
- [DSB12] is another individual submission draft that defines the API for the decoupled streams mechanism, which is mandatory for an application to make use of predefined stream mapping.
- [ABD+12] is the result of a collaboration with the developers of CMT-SCTP (see [IAS06]) and non-renegable selective acknowledgements (see [NEY+08]). It contains all mechanisms for efficient CMT-SCTP usage over dissimilar paths with a fair behaviour on shared bottlenecks. In particular, this includes the mechanisms of buffer splitting, chunk rescheduling and the congestion controls CMT/RPv1, CMT/RPv2 as well as the MPTCP-like congestion control. While this draft is currently still an individual submission, a discussion of the document at the 81st IETF meeting in July 2011 has reached a consensus that this work should be forwarded. That is, it is assumed that the document will become a working group draft in the very near future.

An overview of the ongoing standardisation process for SCTP as well as a survey of the current research on CMT-SCTP has also been published in [DRS+11].

9.2 Outlook and Future Work

While this thesis has evaluated a lot of important and interesting aspects of CMT-SCTP, there are still many more things to be analysed in detail as part of future work – particularly also in the ongoing DFG project on SCTP.

Clearly, for the performance on dissimilar paths, the main problems have been solved now. However, some details should be examined in more depth. Particularly, future work will analyse the performance for non-saturated transfers with small messages. Non-saturated transfers may lead to application-based bursts, in contrast to Transport Layer bursts which are handled by the “Max Burst” burst mitigation variant. Therefore, a CMT-aware strategy that handles both kinds of bursts but avoids the issues of “Use It or Lose It” and “Congestion Window Limiting” may be useful.

Furthermore, chunk rescheduling needs to be extended to the handling of small messages, which are bundled in packets. Clearly, if one packet is lost or overly delayed, multiple DATA chunks could be missing. Therefore, the chunk rescheduling algorithm may be extended to take bundling behaviour into consideration and select multiple chunks for preventive retransmission within a single packet.

Another interesting topic is the performance optimisation of data structures to handle the acknowledgement mechanism at the sender side. For dissimilar paths, the range between the lowest unacknowledged TSN and the highest TSN in the send buffer may be large. This may lead to a performance bottleneck on dissimilar high-speed paths (e.g. $\gg$100 Mbit/s), as it has been observed during testbed measurements.

Finally, there is still much room for improvements of the congestion control. As currently ongoing work, the three congestion control variants introduced in this thesis have been implemented into FreeBSD kernel SCTP and are now evaluated in the testbed environment. An interesting observation for the congestion controls has been the effect of congestion balancing, which may lead to only a small performance improvement for a CMT-SCTP flow, but to a significant improvement for a concurrent non-CMT flow in certain dissimilar path scenarios. While this effect does not violate the performance goals and may also be intended – from a global view – to achieve a better distribution of bandwidth among the flows of a network, it leads to the demand for a more exact definition of "fairness" in the context of networks with multipath transfer. Obviously, there are different perspectives on fairness, which points out to the necessity of future work to develop more fine-granular and formal definitions of "fairness", in order to allow more sophisticated congestion control variants to provide a better tuning of the fairness performance to the needs of network providers, users and their applications.

Appendix A

Reliable Server Pooling

This appendix shortly introduces the Reliable Server Pooling (RSerPool) framework, which is used by the simulation tool-chain described in Appendix B. A more detailed introduction is e.g. provided in [Dre11, DR09, Dre07].

A.1 Architecture

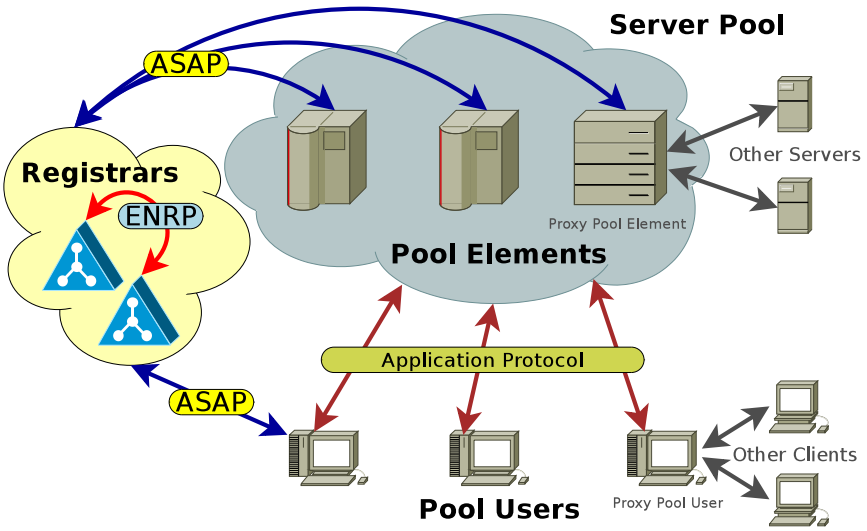


Figure A.1: The Reliable Server Pooling Architecture

RSerPool provides a lightweight framework for the management of server pools and sessions with these server pools, in order to support availability-critical applications as well as load balancing. Figure A.1 depicts an overview of the RSerPool architecture, as defined by [LOTD08] as RFC 5351, with its three types of components:

Pool Element (PE) denotes a server in a pool. PEs in the same pool provide the same service.

Pool User (PU) denotes a client that uses the service of a pool.

Pool Registrar (PR) – shortly denoted as *Registrar* – is the management component for the pools.

Each pool is identified by a unique *Pool Handle* within its so-called *Handlespace*. The handlespace denotes the set of all pools within an *Operation Scope*, which is the administrative domain of the RSerPool setup. For example, this could be an organisation or a department.

The RSerPool architecture also provides a migration path for existing, non-RSerPool applications: *Proxy Pool Users* connect non-RSerPool clients to a server pool; *Proxy Pool Elements* let non-RSerPool servers join a pool.

A.2 Registrar Operations

In order to avoid a PR becoming a *Single Point of Failure*, which leads to a service disruption upon its failure, at least two PRs have to be deployed. The *Endpoint haNdlespace Redundancy Protocol* (ENRP, defined in [XSS+08, DZ12b]), takes care of synchronising a handlespace among the PRs of an operation scope. SCTP is used as the underlying Transport Layer protocol.

In contrast to Grid Computing as introduced by [Fos02, FKNT02], an operation scope is restricted to a single administrative domain. That is, all of its components are under the control of the same authority. This property leads to small management overhead; details are described by [DR08b]. In particular, it also allows for RSerPool usage on devices having only limited memory and CPU resources (e.g. telecommunications equipment). Nevertheless, components may be distributed globally, in order to continue their service even in case of "localised disasters" – like a flooding or an earthquake – as described by [DR07].

A.3 Pool Element Operations

PEs may use an arbitrary PR of their operation scope to register into a pool, by using the Aggregate Server Access Protocol (ASAP, defined in [SXST08a, Dre12c]) which is also transported over SCTP. Upon registration at a PR, the chosen PR becomes the *Home Pool Registrar* (PR-H) of the newly registered PE. A PR-H is responsible for monitoring the availability of its PEs by keep-alive messages, which have to be acknowledged by a PE within a configured timeout. Furthermore, the PR-H is responsible for propagating the information about its PEs to the other PRs of the operation scope. PEs re-register regularly and for information updates; they may also intentionally deregister from their pool. Similar to a registration and re-registration, the PR-H is also responsible for making a deregistration known within its operation scope.

A.4 Pool User Operations

In order to access the service of a pool, a PU requests a PE selection – in form of a so-called *Handle Resolution* – from an arbitrary PR of the operation scope, again by using ASAP. The chosen PR selects the requested list of PE identities and returns them to the PU. The pool-specific selection rule is denoted as *Pool Member Selection Policy* or – in short – *Pool Policy*. Two classes of load distribution policies are supported: non-adaptive and adaptive strategies. While adaptive strategies base their selections on the current PE state (requiring up-to-date state information), non-adaptive algorithms do not need such data. Adaptive and non-adaptive pool policies are defined in [DT08, DZ12a]; a detailed overview is provided by [Dre07, DR05, DR08b].

A further responsibility of ASAP is to let PUs report the unreachability of PEs. A PR counts such unreachability reports and may remove a PE identity from the handlespace when a predefined

threshold has been reached. This mechanism can ensure that "dead" entries are quickly removed from the handlespace, without the need for an overly high (and therefore costly) monitoring interval. More details on this mechanism can be found in [DR09, DZB+10].

Between PU and a pool, the ASAP protocol can provide a Session Layer functionality. That is, a PU establishes a session with a pool, ASAP takes care of selecting a PE, maintaining a Transport Layer connection and triggering a failover to a new PE in case of a failure. This Session Layer functionality is described in more detail, including real-world examples and code listings, by [Dre11, DB10, ZDB+10].

A.5 Automatic Configuration

While it is possible to configure a list of PRs into each RSerPool component, the RSerPool framework provides also the possibility of auto-configuration: PRs may make themselves known within their operation scope by sending so-called *Announces*, i.e. ASAP and ENRP messages which are regularly sent over UDP via IP multicast. The announces of the PRs can be heard by the other components (within the same multicast domain), which can maintain a list of the currently available PRs. That is, RSerPool components are usually just turned on and everything works automatically. More details on this mechanism are described by [Dre11].

A.6 Application Scenarios

Although the standardisation of the RSerPool framework had initially been motivated by the need for availability of SS7 services over IP networks (as for SCTP, see also Subsection 3.14.1), it has been designed for application independence. Research on the applicability and the performance of RSerPool includes use cases like

- battlefield networks (see [UZF+03, UZF+04]),
- e-commerce systems (see [Dre02, DR09]),
- IPFIX (see [DCC12]),
- seamless mobility (see [CJL07, DJT03, DP12]),
- video on demand (see [MR06]),
- VoIP (see [RSB+05, BGPS04, CJR+02]),
- web server pools (see [Dre07]) and
- workload distribution (see [DR05, DR08a, Dre12b]).

A.7 Summary

This appendix has shortly introduced the RSerPool framework with its components, protocols and core functionalities.

Appendix B

SIMPROCTC – The Simulation Processing Tool-Chain

This appendix describes the simulation tool-chain which has been developed in order to efficiently perform the simulations with the model described in Chapter 5.

B.1 Overview

In order to handle simulation tasks efficiently, the model-independent Simulation Processing Tool-Chain – shortly denoted as SIMPROCTC – has been developed. It has been published in [DZR09, DR08a]; the GPLv3-licensed open source package – including a complete example simulation – can be found on the project web site[1]. Figure B.1 shows an overview of SIMPROCTC:

- At its core, the statistics and data processing package GNU R[2] (described by [R D11]) is applied to parametrise simulation runs (to be explained in Section B.3).
- The runs are processed by using GNU MAKE[3] (documented by [SMS10]) – either on the local machine or in an RSerPool-based computation pool (to be explained in Section B.3).
- The post-processing stage prepares the results for their visualisation (to be explained in Subsection B.4.1) as documents in the *Portable Document Format* (PDF; see [Ado08]).
- Optionally, SENDXMPP[4] may be applied to send an instant message to the user on starting as well as on finishing the processing and post-processing of a simulation, by using XMPP (see Subsection 2.13.4). This feature is particularly convenient in combination with an XMPP client on a smartphone, in order to get a nearly immediate notification when the results of a long-running simulation are ready.

Up until now, SIMPROCTC has been successfully deployed for research on SCTP (e.g. documented in [DBAR11b, DRS+11, ADB+11, DBRT10, DSTR10, DBPR10a]), RSerPool (e.g. documented in [DZB+10, DR09, SDR08, DRZ08]) and QoS (e.g. documented in [ZDRZ09, ZDR08]).

[1] SIMPROCTC: http://www.iem.uni-due.de/~dreibh/omnetpp/.
[2] GNU R: http://www.r-project.org/.
[3] GNU MAKE: http://www.gnu.org/software/make/.
[4] SENDXMPP: http://sendxmpp.platon.sk/.

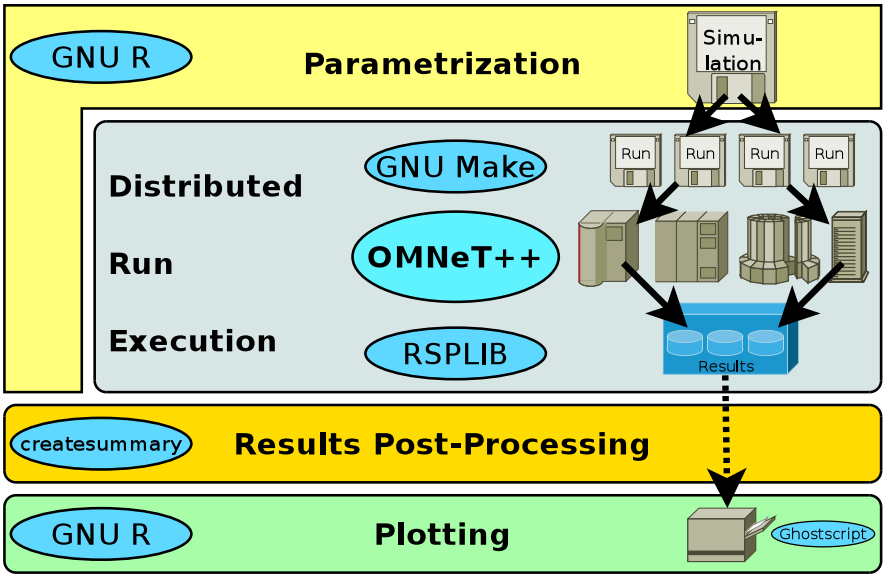


Figure B.1: An Overview of the SIMPROCTC Tool-Chain

B.2 Simulation Parametrisation and Processing

The first step of performing an OMNET++ simulation run is to parametrise the model by writing an appropriate .ini file (see also Section 5.2). The core of SIMPROCTC is therefore a GNU R script which is responsible for performing this simulation parametrisation task.

B.2.1 Formal Definitions

To describe the simulation parametrisation, the introduction of some formal definitions is necessary first: let a simulation model have n *Parameters* $p_1, \ldots, p_n$; $\hat{P}_1, \ldots, \hat{P}_n$ are the corresponding *Parameter Spaces* which contain all possible values. That is, $p_i \in \hat{P}_i$ for all $i \in \{1, \ldots, n\}$. Then, the *Model Parameter Space* is $\hat{P} = \hat{P}_1 \times \hat{P}_2 \times \ldots \times \hat{P}_n$ (i.e. the Cartesian product of the parameter spaces). Using the definition of the model parameter space $\hat{P}$, a *Simulation* $S \subset \hat{P}$ simply contains all parameter combinations $s \in S$ for which a *Run* has to be performed.

Note, that the run number – which corresponds to a certain seed for the random number generator – is simply another input parameter. The simulation executable itself constitutes a *Simulation Function* f, mapping the simulation S to a *Result Space* R:

$$\begin{aligned} f &: S \to R \\ s &\mapsto f(s) \end{aligned}$$

That is, $s \in S$ is mapped to a result $f(s) \in R$; the result consists of vectors and scalars (see Section 5.2; a formal definition is omitted here). It is furthermore assumed that the same setting of s generates the same – or a comparable – output. At least, differences must not falsify the results. For example, the CMT-SCTP model writes the actual run execution real-time as a scalar to allow for profiling (which, of course, depends on the processing system and its current load).

Listing 5 An Example Simulation Configuration in the CMT-SCTP Environment

```
1  simulationConfigurations <- list(
2     # ====== Network QoS ettings ======================================
3     # ------ Northern Trail ------------------------------------
4     list("alphaTrailDataRate",              100000),
5     # ------ Southern Trail ------------------------------------
6     list("betaTrailDataRate",               10, 100, 1000, 5000, 10000,
7                                             15000, 20000, 25000, 30000,
8                                             35000, 40000, 45000, 50000,
9                                             55000, 60000, 65000, 70000,
10                                            75000, 80000, 85000, 90000,
11                                            95000, 100000),
12
13    # ====== Application Parameters ====================================
14    list("measurementDuration",             60),
15    list("unordered",                       1.0),
16
17    # ====== SCTP Parameters ===========================================
18    list("cmtCCVariant",                    "cmt", "cmtrp", "cmtrpv2"),
19    list("cmtBufferSplitVariant",           "none", "bothSides"),
20    list("nrSack",                          "false", "true"),
21    list("queueSize",                       250000),
22    list("arwnd",                           125000)
23
24    ...
25 )
```

B.2.2 Realisation

The parametrisation of a simulation S – which clearly consists of defining the model parameter space subset S – is performed by a GNU R script, as shown in the example in Listing 5. The variable storing S is called simulationConfigurations; it is a list containing sub-lists. Each sub-list includes the parameter name (as first item) and all values to be used (as further items). That is, e.g. line 18 defines three settings of the parameter cmtCCVariant: "cmt", "cmtrp" and "cmtrpv2".

From the settings stored in simulationConfigurations, .ini files are created and finally processed by the simulation model. These steps have to meet the following two goals, in order to achieve an appropriate level of efficiency:

Extensibility It must be possible to add further values for some parameters, but without having to reprocess already performed simulation runs.

Parallelism Simulation runs should be processed in parallel – either on the same system (i.e. on multi-CPU and/or multi-core machines) or on different systems.

To achieve the first goal, the GNU R-based simulation script – denoted as `simulation.R` – first creates a separate *Run Directory* for each run $s \in S$. The initial version of SIMPROCTC has named the directory using a textual representation of s. Clearly, with a rising number of parameters, this approach has easily reached the path length limit of the system[5]. Furthermore, directory names

[5]The limit on Linux systems is e.g. 4,032 bytes if using the REISERFS file system, but only 255 bytes for EXT3.

Listing 6 The Defaults Specification for the CMT-SCTP Environment

```
sctpDefaultConfiguration <- list(
   # ====== SCTP Parameters ==========================================
   list("cmtCCVariant",                "off"),
   list("cmtCUCVariant",               "pseudoCumAckV2"),
   list("cmtUseSFR",                   "true"),
   list("cmtUseDAC",                   "true"),
   list("cmtUseFRC",                   "true"),
   list("cmtBufferSplitVariant",       "none"),
   list("nrSack",                      "false"),
   ...

   ...
)
```

requiring several screen lines to be printed were *really* unhandy. The solution has been to use a Secure Hash Algorithm No. 1 (SHA-1, see [EJ01]) hash value over s as name base instead. This results in appropriately small and usable directory names.

For each run s, a separate .ini file is generated by `simulation.R` in the corresponding run directory. It also specifies its own scalar and vector files, which will be placed in the same directory. A model-specific function writes the parameter settings into the .ini file. That is: for each $s \in S$, the core script sets GNU R variables – the names are given by the parameter names – to the actual values defined by s. Then, a model-specific function called simCreatorWriteParameterSection() writes them as parameters into the .ini file. For the example shown in Listing 5, the parameter cmtCCVariant defines the configuration of CMT for the CMT-SCTP model. The network setup is as illustrated in Figure 5.2 and described in Section 5.3, i.e. the actual instances of the *StandardHost* module including an SCTP instance are part of a compound module array core. Therefore, the parameter lines written into the .ini file e.g. look like: cmttest4.core[0].serverMain[0].sctp.cmtCCVariant="cmt". The usage of GNU R script code in simCreatorWriteParameterSection() to actually write the .ini file allows more sophisticated parametrisation – like writing multiple entries or even computing the actual values to be used in the .ini file – as well.

Furthermore, `simulation.R` creates a Makefile for GNU MAKE – which is described in detail by [SMS10] – for the whole simulation S. Each $s \in S$ leads to an entry performing the following tasks:

1. Old output files (i.e. vector, scalar and log files) are removed.

2. The simulation model executable is run, parametrised by the corresponding .ini file for s. A log file of the run (containing text output for debugging purposes) as well as scalar and vector files are written.

3. All output files are compressed by using the BZIP2 compression tool (described in detail by [Sew07]). This achieves a significant disk space gain, since the output files contain plain text.

4. Finally, a time-stamp file – denoted as *Status File* – is written after successfully processing all former steps. Its existence tells GNU MAKE that the corresponding run has been completed successfully.

A further run of the simulation script will update existing status files by default. That is, already executed runs will not be reprocessed again – since their result would not change (due to the assumption for f in Subsection B.2.1). If the simulation function f changes, this update step can be skipped. Then, all runs will be processed again.

Run directories remain stored until they are deleted manually. This realises the desired caching behaviour: if the simulation is modified from S to $S' \subset \hat{P}$, $S' \neq S$, it is only necessary to process the new runs $s \in S' \setminus S$. The runs $\overline{s} \in S \setminus S'$ remain stored and may be reused after further modification of the simulation, e.g. after having made a couple of runs with a reduced number of parameter settings.

B.2.3 Handling Model Enhancements

A common task during simulation-based research is to extend the functionalities of an existing simulation model. That is, new functionalities – and therefore new parameters – are added. Obviously, these new parameters have to be set to perform runs of the new model. In the usual case, the new functionalities and behaviours of the model can be turned on by some parameter settings (e.g. the CMT-SCTP model has been enhanced by support for NR-SACK). Having already created a set of simulations by using the parametrisation approach described in Subsection B.2.2, this leads to a problem: simply running these scripts results in the lack of parameter specifications for the new functionalities. That is, it would be necessary to modify all of these scripts to set the new parameters appropriately, in order to turn the new functionalities off and retain the old behaviour of the model (e.g. not to use NR-SACK).

The approach for this challenge is straightforward: a global *Default Configuration* D for the simulation environment is specified – in the same way the simulation S is defined. Listing 6 shows an example for the CMT-SCTP environment. Each time a new parameter is added to the model, a default for the new parameter is set here. An obviously useful setting is to turn the new functionality off by default, i.e. the model behaves as before the change. The default configuration must contain exactly one value for each parameter.

The following four *Merging Rules* are applied, in order to generate the actual simulation S^* from a simulation definition S and default settings D:

1. It is allowed that some parameter values are not specified in S. Formally, this could be reached by having each parameter space $\hat{P}_i$ containing an "undefined" entry $\varnothing$.

2. If a certain parameter p_i is not specified in S, the corresponding value is taken from the default settings D. That is: if there is no setting, the default for p_i is used.

3. Else, if there is a setting for a parameter p_i, it is used and the default value is simply ignored.

4. If there is a parameter setting in S but no default in D, the simulation script stops with an error message. Since D contains a default for every parameter of the model, this cannot happen in theory. However, this kind of error may be caused by typos in the parameter specification of S. Avoiding this kind of problem is therefore highly useful in practise.

An example is provided by the simulation configuration $\mathrm{simulationConfigurations}$ S in Listing 5 and the default configuration $\mathrm{sctpDefaultConfiguration}$ D in Listing 6: according to Rule #3, the parameter values for $\mathrm{cmtCCVariant}$ are taken from S (i.e. "cmt", "cmtrp", "cmtrpv2"). The simulation parameter nrSack is not defined in S. It is therefore taken from the defaults D (i.e. using the default value "false"), according to Rule #1 and Rule #2. If there would be a parameter $\mathrm{someNonExistingParameter}$ in S – which is not defined in the defaults D – this would cause an error, according to Rule #4.

Another advantage of the default configuration mechanism is that the actual simulation configuration S may remain small. In a usual simulation setup, there are only a few parameters which actually have to be modified, while most parameters just use their default values. In particular, this keeps the simulation file for S also easily understandable, e.g. by users having only limited knowledge of the full set of configurable parameters provided by a complex simulation model. For example, the configuration in Listing 5 is nearly complete, although more than 100 parameters are provided by the CMT-SCTP simulation environment.

B.3 Distributed Simulation Processing

As last step, `simulation.R` executes GNU MAKE to actually perform the simulation runs.

B.3.1 Overview

In order to improve simulation processing performance, `simulation.R` already obtains the number of CPUs/cores[6] and lets GNU MAKE execute an appropriate number of runs in parallel[7]. That is, a dual-core machine should perform two runs simultaneously. Nevertheless, all runs are processed by only a single PC.

In order to allow parallel simulation processing in a PC pool, some solutions like AKAROA (see [Var10, Section 9.5]) had been considered first. However, the configuration in the quite heterogeneous network (different Linux versions, different subnets, downtime when PCs are used for student exercises or projects, etc.) had been challenging and a "lightweight" approach for simulation distribution had been desired. Since RSerPool is a lightweight framework for server pool management and workload distribution (see Appendix A), it has therefore been quite straightforward to utilise RSerPool for this task.

B.3.2 The Scripting Service

The RSPLIB[8] package is a GPLv3-licensed, open source implementation of the RSerPool framework. It has been developed as part of the Ph.D. project by [Dre07] at the University of Duisburg-Essen, a detailed introduction is also provided in [Dre11, Dre05, DT03]. The *Scripting Service* (SS), which is included in the RSPLIB package as example application, provides workload distribution functionality: the user of this service provides

- a TAR/GZIP-archived *Work Package* and
- optionally a TAR/BZIP2-archived *Work Environment*

to the scripting service PU. The PU creates a session with the scripting service pool and provides the work package to a scripting service PE. Also, if the PE does not yet contain the work environment, it is also transferred to the PE. The PE stores the work environment in its *Environment Cache*; further work packages for the same work environment do not need another transfer. The differentiation between work environments is realised by an SHA-1 hash value (see [EJ01]) computed over the work environment data. Work environment and work package are extracted into a temporary directory. Then, a script named `ssrun`, which is included in the work package, is executed. This script may write an

[6] By using the CPU information from `/proc/cpuinfo`.
[7] By using the GNU MAKE parameter `-j [jobs]`.
[8] RSPLIB: http://tdrwww.iem.uni-due.de/dreibholz/rserpool/.

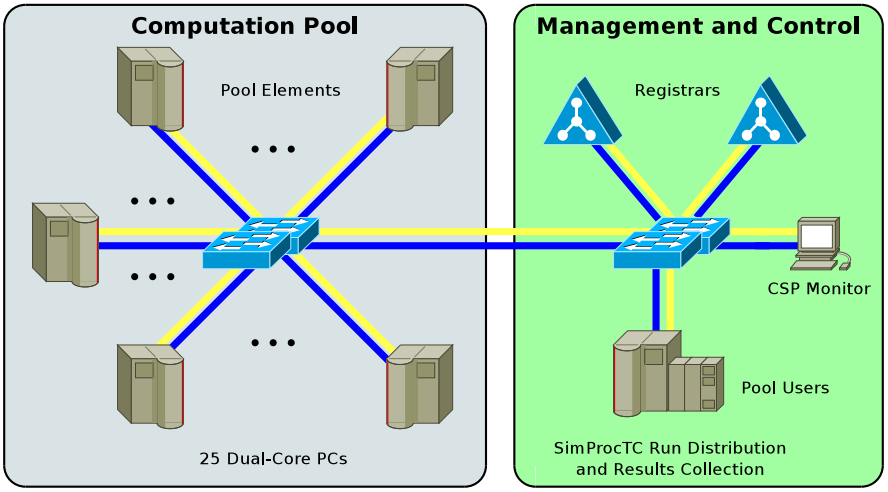


Figure B.2: The Scripting Service Pool used for the Simulation Computation

output archive, which is finally downloaded by the PU. The protocol used for communication between PU and PE is denoted as the *Scripting Service Protocol* (SSP); like other RSerPool communications, it uses SCTP for transport.

The scripting service pool applies the adaptive "Least Used" policy for PE selection (for details, see also [DR09, DR08b, DR05]): the least-loaded PE is used, with round-robin selection among equally loaded PEs. Each PE can handle up to SSMaxThreads sessions simultaneously; the load value is set according to the currently handled number of sessions. By default, SSMaxThreads is set to the number of CPUs/cores of the PC hosting the scripting service PE.

If a PE rejects a session (since it is already serving SSMaxThreads sessions), or if it goes out of service (e.g. the PC is turned off), the session is simply restarted from scratch ("Abort and Restart" principle) after a short delay (e.g. 5 s). This delay avoids overloading the network with reject-and-retry floods (see also [ZDR07]) when there are too few PEs available.

Applying the scripting service of RSPLIB for the simulation processing is easy: the simulation model executable as well as all files to run it, which particularly includes all used shared libraries as well as the shared library loader itself, are packaged into a work environment archive. Furthermore, instead of invoking the simulation model executable from the Makefile by GNU MAKE itself, a script denoted as `ssdistribute` is called. This script simply archives the .ini file and a script called `ssrun` into a work package archive and provides it, together with the work environment, to the scripting service PU. The PU distributes this archive to a PE in the scripting service pool, and the PE executes `ssrun`. The `ssrun` script actually calls the simulation model executable, collects scalar, vector and log files and puts them together into an archive. This archive is downloaded by the PU and finally extracted by `ssdistribute` into the corresponding run directory.

B.3.3 The Pool Setup

Figure B.2 illustrates the scripting service pool setup of the networking lab at the Institute for Experimental Mathematics of the University of Duisburg-Essen. It consists of 25 PEs running either Kubuntu Linux 10.10 ("Maverick Meerkat") or FreeBSD 8.2, with all of them having a dual-core CPU (either 32 bit or 64 bit), i.e. the pool can process 50 parallel sessions in total. Further PEs may

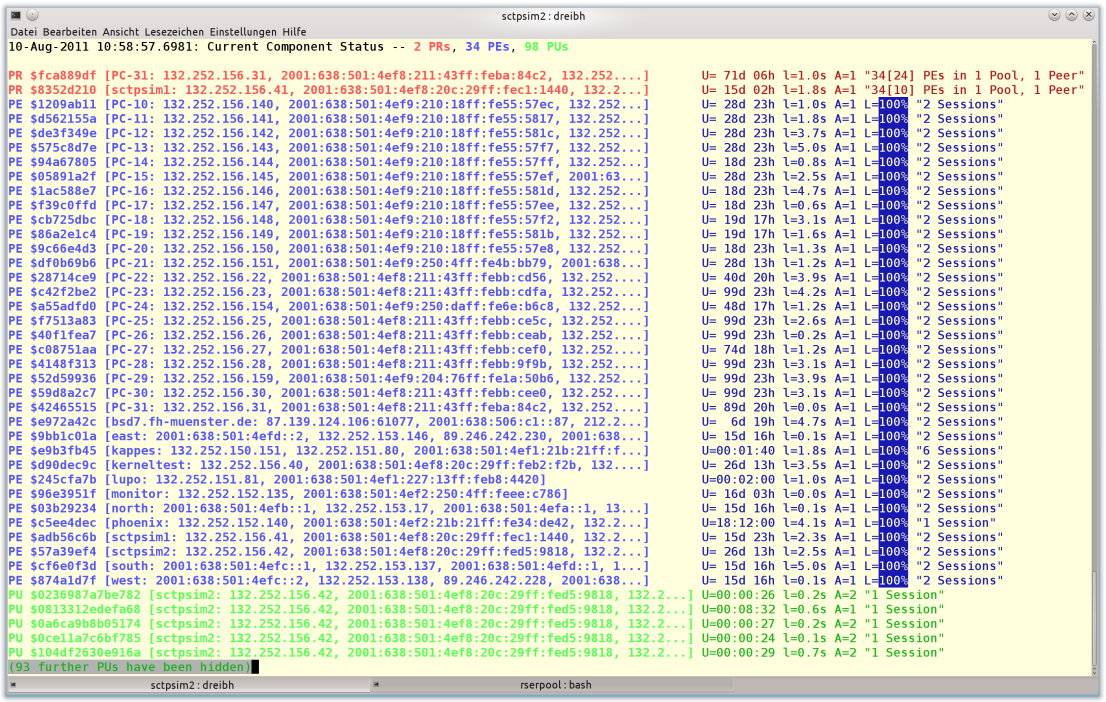


Figure B.3: A Screenshot of the cspmonitor Output

be added temporarily in order to further increase the computing performance of the pool. The lab PCs are connected via two independent Ethernets (blue and yellow), both using IPv4 and IPv6 simultaneously. That is, the systems are quad-homed. Automatic configuration as explained in Section A.5 is applied: each PE will automatically find a PR for registration; two PRs are installed to provide redundancy. PEs may be dynamically added and removed, i.e. when the PCs are temporarily needed for other tasks, the scripting service PEs may be stopped. On the PU side, GNU MAKE has to be called with an appropriate number of simultaneous processes (at least 50 to utilise all 50 cores). Then, there will be up to the given number of parallel simulation run sessions.

B.3.4 The Component Status Protocol

When deploying an RSerPool setup consisting of significantly more than a few components, the scenario becomes confusing quickly. In order to get a complete picture of the current status of all components (i.e. PRs, PEs and PUs), monitoring becomes a handy tool. Monitoring is not a part of RSerPool itself, but RSPLIB provides this feature for all of its applications. It allows them to report their status, including hostname, IP address(es), uptime, load and other useful status information to a central monitoring component. This status reporting is realised by the simple, unidirectional *Component Status Protocol* (CSP; details can be found in [Dre11]), transported over UDP. The simple monitoring application provided by RSPLIB – called `cspmonitor` – displays this information on screen. Figure B.3 shows a screenshot of the output from the lab pool described in Subsection B.3.3, consisting of 2 PRs, 34 fully utilised PEs and 98 PUs (only 5 of them are visible in the screenshot). Note, that the setup shown here consists of the lab pool described in Subsection B.3.3 (25 PEs) which has been extended by 9 additional PEs.

B.3.5 The Scripting Service Pool – A Stress Test for SCTP Implementations

Besides the usefulness of the scripting service setup to provide a distributed processing of simulation runs, the scenario has also another important function: it provides a relatively hard stress test for the SCTP implementations involved: a separate session, including association establishment and bidirectional data transfer between quad-homed endpoints with simultaneous IPv4 and IPv6 usage, is used for each simulation run. Furthermore, error cases – like unplugged cables, reconfigured interfaces, etc. – occur frequently, since the lab PCs are also used for student exercises.

As result of these tests, multiple problems have been found and fixed. The most important issue has been a bug[9] in the Linux kernel SCTP implementation, which has been documented as CVE-2010-3432[10] (Common Vulnerabilities and Exposures). This bug was remotely exploitable for causing a denial of service attack on Linux systems with a wide set of affected kernel versions.

B.4 Results Post-Processing and Visualisation

When all simulation runs have eventually been processed, the results have to be visualised for analysis and interpretation. SIMPROCTC provides support to visualise the scalar results, which are distributed over the scalar files located in the run directories. Therefore, the first step necessary is to bring the data from the various scalar files into an appropriate form for further post-processing. This step is denoted as *Summarisation*; an introduction is also provided in [DZR09].

B.4.1 Scalars Summarisation

The summarisation task is performed by a C++-written program named `createsummary`. An external program – instead of just using GNU R itself to perform this step – is used due to the requirements on memory and CPU power. `createsummary` is called as last step of the Makefile and iterates over all scalar files of a simulation S. Each file is read – with on-the-fly BZIP2-decompression – and each scalar value as well as the configuration $s \in S$ having led to this value – are stored in memory. Depending on the number of scalars, the required storage space may have a size of multiple GiB.

Since usually not all scalars of a simulation are required for analysis (e.g. for an SCTP simulation, it may be unnecessary to include IP or PPP statistics), a list of scalar name prefixes to be excluded from summarisation can be provided to `createsummary`, in form of the so-called *Summary Skip List*. This feature may significantly reduce the memory and disk space requirements of the summarisation step. Since the skipped scalars still remain stored in the scalar files themselves, it is possible to simply re-run `createsummary` with updated summary skip list later, in order to also include them.

Having all relevant scalars stored in memory, a data file – which can be processed by GNU R or other programs – is written for each scalar. The data file is simply a table in text form, containing the column names on the first line. Each following line contains the data, with line number and an entry for each column (all separated by spaces); an example is provided in [DZR09, Listing 3]. That is, each line consists of the settings of all parameters and the resulting scalar value. The data files are also BZIP2-compressed on the fly, in order to reduce the storage space requirements.

[9] Bug in sctp_packet_config(): http://marc.info/?l=linux-netdev&m=128453869227715&w=3.
[10] CVE-2010-3432: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3432.

B.4.2 Plotting

The results of a simulation may finally be presented visually by plotting the data from the output files written by `createsummary` or alternatively other output data (particularly, the vectors). Since GNU R – as described in [R D11] – also contains a rich set of graphics functions, and it has already been used for simulation parametrisation, it has therefore been straightforward to also apply it for plotting the results. In particular, it allows a very fine-granular control of the output plots to adapt the presentation to special requirements (e.g. labels, grids, colours, line styles, etc.). Nevertheless, of course, it would also be possible to apply any other tool – e.g. GNU PLOT[11], GNU OCTAVE[12], LIBREOFFICE[13] or even MICROSOFT OFFICE[14] – for visualising the results.

For result analysis, it is crucial that the impact for variations of multiple parameters can be presented in an easy-to-understand form. Therefore, the plotting script – named `plotter.R` – applies the idea of mapping parameters and the result to axes of the plot:

- The main parameter is displayed on the X-axis, the result (e.g. the value of a scalar) on the Y-axis.
- There may be multiple lines per plot: the Z-axis identifies the line. For readability, a different colour or shade (on a grey-scale plot) is used for each line.
- The Z-axis may be further subdivided by V- and W-axes: the V-axis uses a different line style; the W-axis uses a different point style.
- On one page, there may be multiple plots: the A-axis divides the page in horizontal direction, the B-axis in vertical direction (i.e. there is a separate plot for each A/B-axis value).
- Finally, the P-axis creates a new page for each P-axis value; each of these pages uses a distinct background colour (on a colour plot).

To make the plot functionality clearer, Figure B.4 shows an example plot from a CMT-SCTP simulation in a dual-homed environment. Here, the emphasis is on the *presentation* of the results, not their actual meaning (the explanations can be found in Chapter 7 and Chapter 8). The X-axis presents the bandwidth of Path #2 – denoted as ρ_2 and the Y-axis the payload throughput achieved by NETPERFMETER. The usage of buffer splitting Π, with two variants marked by two different colours (Π=bothSides in orange colour; Π=none in cyan colour), is depicted by the Z-axis. Furthermore, the V-axis represents the support of NR-SACK ν, with two variants marked by two different line styles (ν=true as solid line; ν=false as dashed line).

The A-axis displays the congestion control variant Γ, with its three variants (Γ=cmt, Γ=cmtrp, Γ=cmtrpv2); the B-axis shows the usage of outstanding bytes for Buffer Splitting Ω, with the two possible variants (Ω=true, Ω=false). The values for A-axis and B-axis are displayed in different-coloured boxes, which provides a good differentiability. Only one P-axis value is displayed here: a Path #1 bandwidth of ρ_1=100 Mbit/s (this value is shown below the plot title). The second P-axis value is neglected here, due to space limitations. It would just be another plot page, with a different background colour.

Further features of the `plotter.R` script are:

[11] GNU PLOT: http://www.gnuplot.info/; see also [Cra10].
[12] GNU OCTAVE: http://www.gnu.org/software/octave/; see also [Eat11].
[13] LIBREOFFICE http://www.libreoffice.org/.
[14] MICROSOFT OFFICE: http://office.microsoft.com.

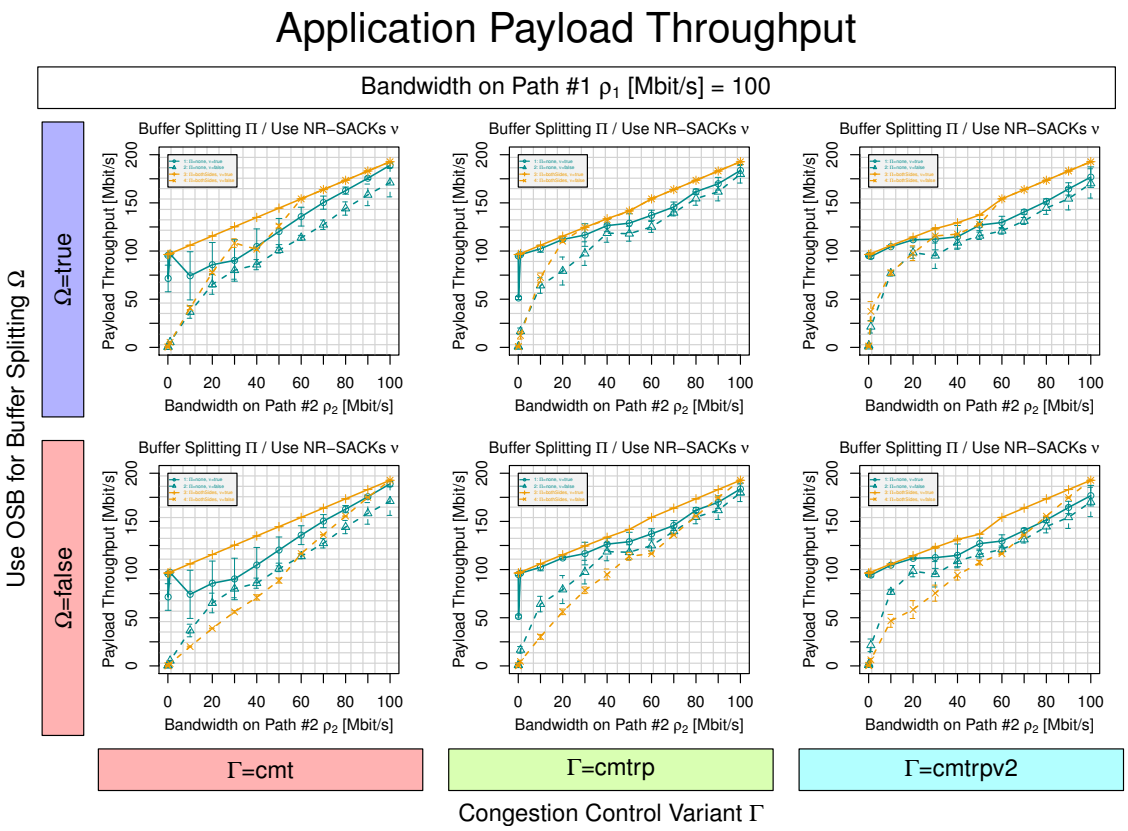


Figure B.4: A Complex Example Plot using X/Y/Z/V/A/B/P Axes

- The output can optionally be in black and white, grey scale or colour.
- A mapping from an axis label to a variable may be defined to simplify writing a description (e.g. $\Gamma \rightarrow$ Congestion Control Variant).
- If there are multiple values per plot point (e.g. from runs with different seeds), the average value is taken for plotting. Furthermore, the confidence intervals (usually 95%) are computed and displayed.
- The plots are written as PDF files (i.e. in a scalable vector graphics format; see also [Ado08]), for inclusion into pdfLaTeX documents. If necessary, these PDF files may be converted into raster formats like PNG (see Subsection 2.13.4), e.g. for inclusion into LibreOffice or Microsoft Office.
- Finally, the resulting PDF files may be processed by GhostScript[15], in order to embed all required fonts. This avoids display problems when processing the files on different platforms (particularly, on a printer). Since this step also results in a compression of the files, their size is furthermore reduced.

[15] GhostScript: http://www.ghostscript.com/.

B.4.3 Plotting Templates

If there is a larger number of plots to be created, e.g. as for this thesis, the plot definition task becomes increasingly time-consuming. Therefore, the SIMPROCTC `plotter.R` script provides the feature of so-called *Plot Templates*, which may be applied to actually define plots. That is, such a template specifies an axis input (e.g. a scalar data file created by `createsummary`), a possible manipulation function (e.g. to convert bit/s to Mbit/s), a label (e.g. "Payload Throughput [Mbit/s]") and a grid colour (e.g. to give different Y-axis results – like average throughput and average delay – distinct colours to enhance readability).

Then, defining the actual plot just means to map templates to the axes (X, Y, Z, V, W, A, B and P) and setting a plot title (e.g. "Application Payload Throughput"). Optionally, ranges for the X- and Y-axes (e.g. from 0 to 100 in steps of 10) and the sorting orders of the axes' values (ascending or descending) may be configured. Finally, a filter may be defined which skips certain results, e.g. to only show results where CMT has been turned on, while the underlying data also contains results for having this option turned off. A detailed example – including an example script – is provided in [DZR09, Subsection 5.3].

B.5 Summary

In this appendix, the SIMPROCTC simulation processing tool-chain has been introduced. It supports the tasks of

- Parametrising a simulation,
- Distributing the simulation runs in an RSerPool-based computation pool by using the scripting service application,
- Collecting and post-processing the results as well as
- Visualising the results.

Particularly, by making use of the SCTP-based RSerPool framework with the scripting service, this application has also been a very interesting stress test for a larger-scale deployment of SCTP. These stress tests have influenced the research on SCTP by triggering some ideas for further performance enhancements.

List of Figures

List of Listings

List of Tables

Bibliography

[AB05] Mark Allman and Ethan Blanton. Notes on Burst Mitigation for Transport Protocols. *ACM SIGCOMM Computer Communication Review*, 35(2):53–60, 2005. ISSN 0146-4833. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.9560&rep=rep1&type=pdf, doi:10.1145/1064413.1064419. 60, 82, 104, 142, 173

[ABD+12] Paul D. Amer, Martin Becke, Thomas Dreibholz, Nasif Ekiz, Janardhan R. Iyengar, Preethi Natarajan, Randall R. Stewart, and Michael Tüxen. Load Sharing for the Stream Control Transmission Protocol (SCTP). Internet Draft Version 04, IETF, Network Working Group, March 2012. draft-tuexen-tsvwg-sctp-multipath-04.txt, work in progress. Available from: http://tools.ietf.org/id/draft-tuexen-tsvwg-sctp-multipath-04.txt. 63, 174

[ADB+11] Hakim Adhari, Thomas Dreibholz, Martin Becke, Erwin Paul Rathgeb, and Michael Tüxen. Evaluation of Concurrent Multipath Transfer over Dissimilar Paths. In *Proceedings of the 1st International Workshop on Protocols and Applications with Multi-Homing Support (PAMS)*, pages 708–714, Singapore, March 2011. ISBN 978-0-7695-4338-3. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/PAMS2011.pdf, doi:10.1109/WAINA.2011.92. 105, 108, 113, 172, 181

[Ado08] Adobe. Portable Document Format – Part 1: PDF 1.7. ISO 32000-1, International Standards Organisation, July 2008. Available from: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf. 181, 191

[Aig99] Martin Aigner. *Diskrete Mathematik*. Friedrich Vieweg & Sohn, Wiesbaden/Germany, 2nd edition, June 1999. ISBN 978-3528172688. 5

[AKO97] Mark Allman, Hans Kruse, and Shawn Ostermann. An Application-Level Solution to TCP's Satellite Inefficiencies. In *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*, Rye, New York/U.S.A., November 1997. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.52.2516&rep=rep1&type=pdf. 69

[AKZ99a] Guy Almes, Sunil Kalidindi, and Matthew J. Zekauskas. A One-way Delay Metric for IPPM. Standards Track RFC 2679, IETF, September 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2679.txt. 22

[AKZ99b] Guy Almes, Sunil Kalidindi, and Matthew J. Zekauskas. A One-way Packet Loss Metric for IPPM. Standards Track RFC 2680, IETF, September 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2680.txt. 23

[AKZ99c] Guy Almes, Sunil Kalidindi, and Matthew J. Zekauskas. A Round-trip Delay Metric for IPPM. Standards Track RFC 2681, IETF, September 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2681.txt. 23

[All03] Mark Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465, IETF, February 2003. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3465.txt. 38

[All07] Allied Telesis. *DSL White Paper*, January 2007. Available from: www.alliedtelesis.com/media/pdf/dsl_wp.pdf. 42

[AM10] Murray Altheim and Shane McCarron. XHTML 1.1 – Module-based XHTML. Recommendation, W3C, November 2010. Available from: http://www.w3.org/TR/2010/REC-xhtml11-20101123/xhtml11.pdf. 49

[APB09] Mark Allman, Vern Paxson, and Ethan Blanton. TCP Congestion Control. Standards Track RFC 5681, IETF, September 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5681.txt. 35, 38, 48

[ASL03] Ahmed Abd El Al, Tarek Saadawi, and Myang Lee. Load Sharing in Stream Control Transmission Protocol. Internet Draft Version 00, IETF, Individual Submission, May 2003. draft-ahmed-lssctp-00, work in progress. Available from: http://potaroo.net/ietf/all-ids/draft-ahmed-lssctp-00.txt. 2

[BAFW03] Ethan Blanton, Mark Allman, Kevin Fall, and Lili Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. Standards Track RFC 3517, IETF, April 2003. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3517.txt. 35, 39

[BBP88] Robert Braden, David A. Borman, and Craig Partridge. Computing the Internet Checksum. Standards Track RFC 1071, IETF, September 1988. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1071.txt. 25

[BCC+98] Bob Braden, David D. Clark, Jon Crowcroft, Bruce Davie, Stephen E. Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, K. K. Ramakrishnan, Scott Shenker, John Wroclawski, and Lixia Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. Informational RFC 2309, IETF, April 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2309.txt. 23, 151

[BCD+98] David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Services. Informational RFC 2475, IETF, December 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2475.txt. 24

[BDH99] David A. Borman, Stephen E. Deering, and Robert M. Hinden. IPv6 Jumbograms. Standards Track RFC 2675, IETF, August 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2675.txt. 48

[BDRF11] Martin Becke, Thomas Dreibholz, Erwin Paul Rathgeb, and Johannes Formann. Link Emulation on the Data Link Layer in a Linux-based Future Internet Testbed Environment. In *Proceedings of the 10th International Conference on Networks (ICN)*, pages 92–98, St. Maarten/Netherlands Antilles, January 2011. ISBN 978-1-61208-002-4. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/ICN2011.pdf. 100, 101, 171

[BFH03] Robert Braden, Ted Faber, and Mark Handley. From Protocol Stack to Protocol Heap – Role-Based Architecture. *ACM SIGCOMM Computer Communication Review*, 33:17–22, January 2003. ISSN 0146-4833. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1741&rep=rep1&type=pdf, doi:10.1145/774763.774765. 15

[BGPS04] Marjan Božinovski, Liljana Gavrilovska, Ramjee Prasad, and Hans-Peter Schwefel. Evaluation of a Fault-Tolerant Call Control System. *Facta Universitatis Series: Electronics and Energetics*, 17(1):33–44, 2004. ISSN 0353-3670. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.5766&rep=rep1&type=pdf. 179

[BIP06] BIPM. *Le Système international d'unités – The International System of Units*. STEDI Media, Paris/France, 8th edition, May 2006. ISBN 92-822-2213-6. Available from: http://www.bipm.org/utils/common/pdf/si_brochure_8.pdf. 20, 22

[Bla07] Eugene Blanchard. *Introduction to Data Communications*. Southern Alberta Institute of Technology, Calgary/Canada, 2.1 edition, January 2007. Available from: http://learnat.sait.ab.ca/ict/txt_information/Intro2dcRev2/index.html. 15

[BLC95] Tim Berners-Lee and Daniel W. Connolly. Hypertext Markup Language – 2.0. Standards Track RFC 1866, IETF, November 1995. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1866.txt. 49

[BM02] Randy Bush and David Meyer. Some Internet Architectural Guidelines and Philosophy. Informational RFC 3439, IETF, December 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3439.txt. 15

[BMR07] Andreas Bluschke, Michael Matthews, and Philipp Rietzsch. Führungsposition verteidigt – Für hohe Bandbreiten sind neue xDSL-Generationen erste Wahl. *NET Zeitschrift für Kommunikationsmanagement*, 2007. Available from: http://www.net-im-web.de/pdf/Bluschke-xDSL.pdf. 42

[Bou97] Thomas Boutell. PNG (Portable Network Graphics) Specification. Informational RFC 2083, IETF, March 1997. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2083.txt. 49

[BPB11] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. MultiPath TCP: From Theory to Practice. In *Proceedings of the 10th International IFIP Networking Conference*, pages 444–457, Valencia/Spain, May 2011. ISBN 978-3-642-20756-3. Available from: http://inl.info.ucl.ac.be/system/files/networking-mptcp.pdf. 75

[Bra89] Robert Braden. Requirements for Internet Hosts – Communication Layers. Standards Track RFC 1122, IETF, October 1989. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1122.txt. 14, 34, 36

[Bra96] Scott Bradner. The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force. Informational RFC 2026, IETF, October 1996. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2026.txt. 45

[Bra05a] Scott Bradner. IETF Structure and Internet Standards Process. In *Proceedings of the 64th IETF Meeting*, Vancouver/Canada, November 2005. Available from: http://mmlab.snu.ac.kr/courses/2006_advanced_internet/handout/1.%20IETF%20newcomers.pdf. 45

[Bra05b] Scott Bradner. Intellectual Property Rights in IETF Technology. Informational RFC 3979, IETF, March 2005. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3979.txt. 46

[Bro06] Martin A. Brown. Traffic Control HOWTO, October 2006. Available from: http://linux-ip.net/articles/Traffic-Control-HOWTO/. 23

[Cam03] Peter J. Cameron. Encyclopaedia of DesignTheory: Galois Fields, May 2003. Available from: http://designtheory.org/library/encyc/topics/gf.pdf. 26

[CBH93] Guy Castagnoli, Stefan Bräuer, and Martin Herrmann. Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits. *IEEE Transactions on Communications*, 41(6):883–892, June 1993. ISSN 0090-6778. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00231911, doi:10.1109/26.231911. 27

[CDG06] Alex Conta, Stephen E. Deering, and Mukesh Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. Standards Track RFC 4443, IETF, March 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4443.txt. 48

[Ced08] Per Cederqvist. *Version Management with CVS*, May 2008. Available from: http://ftp.gnu.org/non-gnu/cvs/source/stable/1.11.23/cederqvist-1.11.23.pdf. 90

[CFM99] Rob Coltun, Dennis Ferguson, and John T. Moy. OSPF for IPv6. Standards Track RFC 2740, IETF, December 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2740.txt. 50

[CHW98] Jon Crowcroft, Mark Handley, and Ian Wakeman. *Internetworking Multimedia*. UCL Press, London/United Kingdom, December 1998. Available from: http://www.cl.cam.ac.uk/~jac22/out/mm.pdf. 19

[CJL07] C. S. Chandrashekaran, Walter L. Johnson, and Abhijit Lele. Method using Modified Chord Algorithm to Balance Pool Element Ownership among Registrars in a Reliable Server Pooling Architecture. In *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware (COMSWARE)*, pages 1–7, Bangalore/India, January 2007. ISBN 1-4244-0614-5. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4268132, doi:10.1109/COMSWA.2007.382489. 179

[CJR+02] Phillip T. Conrad, Andreas Jungmaier, Christopher Ross, Woon-Chiat Sim, and Michael Tüxen. Reliable IP Telephony Applications with SIP using RSerPool. In *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II*, volume X, pages 352–356, Orlando, Florida/U.S.A., July 2002. ISBN 980-07-8150-1. Available from: http://www.recursosvoip.com/docs/english/SCI2002_Reliable_IP_Telephony_with_SIP_and_RSerPool_16_07_2002.pdf. 179

[CK74] Vinton G. Cerf and Robert E. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974. ISSN 0090-6778. Available from: http://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf, doi:10.1109/TCOM.1974.1092259. 14, 24, 37

[Cla08] Benoît Claise. Specification of the IP Flow Information Export (IPFIX) Protocol. Standards Track RFC 5101, IETF, January 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5101.txt. 65

[CMQ87] Smoot Carl-Mitchell and John S. Quarterman. Using ARP to Implement Transparent Subnet Gateways. Standards Track RFC 1027, IETF, October 1987. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1027.txt. 101

[CP10] Brian E. Carpenter and Craig Partridge. Internet Requests for Comments (RFCs) as Scholarly Publications. *ACM SIGCOMM Computer Communication Review*, 40:31–33, January 2010. ISSN 0146-4833. Available from: http://www.cs.auckland.ac.nz/~brian/RFCs-CCR-201001.pdf, doi:10.1145/1672308.1672315. 46

[CR09] Marta Carbone and Luigi Rizzo. Dummynet Revisited. Technical report, Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Pisa/Italy, May 2009. Available from: http://www.onelab.eu/images/PDFs/Scientific_papers/OneLab2/20090531-ccr-dummynet_rizzo.pdf. 97, 100

[Cra10] Dick Crawford. *gnuplot 4.4 – An Interactive Plotting Program*, March 2010. Available from: http://www.gnuplot.info/docs_4.4/gnuplot.pdf. 190

[CXSN04] Kai Chen, Yuan Xue, Samarth H. Shah, and Klara Nahrstedt. Understanding Bandwidth-Delay Product in Mobile Ad Hoc Networks. *Computer Communications*, 27(1):923–934, 2004. ISSN 0140-3664. Available from: http://eecs.vanderbilt.edu/people/yuanxue/publication-files/comcom04-chen-understanding.pdf. 34

[DB10] Thomas Dreibholz and Martin Becke. The RSPLIB Project – From Research to Application. Demo Presentation at the IEEE Global Communications Conference (GLOBECOM), December 2010. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/Globecom2010-Demo.pdf. 179

[DBA12] Thomas Dreibholz, Martin Becke, and Hakim Adhari. SCTP Socket API Extensions for Concurrent Multipath Transfer. Internet Draft Version 03, IETF, Network Working Group, March 2012. draft-dreibholz-tsvwg-sctpsocket-multipath-03.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-tsvwg-sctpsocket-multipath-03.txt. 152, 174

[DBAR11a] Thomas Dreibholz, Martin Becke, Hakim Adhari, and Erwin Paul Rathgeb. Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain. In *Proceedings of the 19th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar/Croatia, September 2011. ISBN 978-953-290-027-9. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/SoftCOM2011.pdf. 92, 171

[DBAR11b] Thomas Dreibholz, Martin Becke, Hakim Adhari, and Erwin Paul Rathgeb. On the Impact of Congestion Control for Concurrent Multipath Transfer on the Transport Layer. In *Proceedings of the 11th IEEE International Conference on Telecommunications (ConTEL)*, pages 397–404, Graz/Austria, June 2011. ISBN 978-953-184-152-8. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/ConTEL2011.pdf. 154, 161, 164, 174, 181

[DBHR10] Thomas Dreibholz, Martin Becke, Christian Henke, and Erwin Paul Rathgeb. An Approach for Transferring an End-to-End Transport Service into a Functional Building Block Structure. In *Proceedings of the 5th GI/ITG KuVS Workshop on the Future Internet*, Stuttgart/Germany, June 2010. Available from: http://www.future-internet.org/files/2010/Folien/Abstract_Dreibholz2.pdf. 15, 103

[DBPR10a] Thomas Dreibholz, Martin Becke, Jobin Pulinthanath, and Erwin Paul Rathgeb. Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 312–319, Perth/Australia, April 2010. ISSN 1550-445X. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/AINA2010.pdf, doi:10.1109/AINA.2010.117. 153, 174, 181

[DBPR10b] Thomas Dreibholz, Martin Becke, Jobin Pulinthanath, and Erwin Paul Rathgeb. Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework. In *Proceedings of the 3rd ACM/ICST International Workshop on OMNeT++*, Torremolinos, Málaga/Spain, March 2010. ISBN 978-963-9799-87-5. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/OMNeT__Workshop2010-SCTP.pdf, doi:10.4108/ICST.SIMUTOOLS2010.8673. 64, 70, 81, 83, 85, 90, 105, 171

[DBRT10] Thomas Dreibholz, Martin Becke, Erwin Paul Rathgeb, and Michael Tüxen. On the Use of Concurrent Multipath Transfer over Asymmetric Paths. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Miami, Florida/U.S.A., December 2010. ISBN 978-1-4244-5637-6. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/Globecom2010.pdf, doi:10.1109/GLOCOM.2010.5683579. 105, 108, 113, 121, 126, 129, 132, 172, 181

[DC02] Carlo Demichelis and Philip Chimento. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). Standards Track RFC 3393, IETF, November 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3393.txt. 23

[DCB+02] Bruce Davie, Anna Charny, Jon Bennett, Kent Benson, Jean-Yves Le Boudec, Bill Courtney, Shahram Davari, Victor Firoiu, and Dimitrios Stiliadis. An Expedited For-

warding PHB (Per-Hop Behavior). Standards Track RFC 3246, IETF, March 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3246.txt. 24

[DCC12] Thomas Dreibholz, Lode Coene, and Phillip T. Conrad. Reliable Server Pooling Applicability for IP Flow Information Exchange. Internet Draft Version 13, IETF, Individual Submission, January 2012. draft-coene-rserpool-applic-ipfix-13.txt, work in progress. Available from: http://tools.ietf.org/id/draft-coene-rserpool-applic-ipfix-13.txt. 179

[DG96] L. Peter Deutsch and Jean-Loup Gailly. ZLIB Compressed Data Format Specification version 3.3. Informational RFC 1950, IETF, May 1996. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1950.txt. 26

[DH98] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6). Standards Track RFC 2460, IETF, December 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2460.txt. 24, 47

[DJT03] Thomas Dreibholz, Andreas Jungmaier, and Michael Tüxen. A New Scheme for IP-based Internet Mobility. In *Proceedings of the 28th IEEE Local Computer Networks Conference (LCN)*, pages 99–108, Königswinter/Germany, October 2003. ISBN 0-7695-2037-5. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/LCN2003.pdf, doi:10.1109/LCN.2003.1243117. 179

[dLB06] Cédric de Launois and Marcelo Bagnulo. The Paths towards IPv6 Multihoming. *IEEE Communications Surveys and Tutorials*, 8(2):38–51, 2006. ISSN 1553-877X. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.5621&rep=rep1&type=pdf, doi:10.1109/COMST.2006.315853. 75

[DM09] Thomas Dreibholz and Jaiwant Mulik. Reliable Server Pooling MIB Module Definition. RFC 5525, IETF, April 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5525.txt. 65

[DP88] Willibald Dörfler and Werner Peschek. *Einführung in die Mathematik für Informatiker*. Hanser Fachbuchverlag, Wien/Austria, July 1988. ISBN 978-3446151123. 5

[DP12] Thomas Dreibholz and Jobin Pulinthanath. Applicability of Reliable Server Pooling for SCTP-Based Endpoint Mobility. Internet Draft Version 11, IETF, Individual Submission, January 2012. draft-dreibholz-rserpool-applic-mobility-11.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-rserpool-applic-mobility-11.txt. 179

[DR05] Thomas Dreibholz and Erwin Paul Rathgeb. On the Performance of Reliable Server Pooling Systems. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 30th Anniversary*, pages 200–208, Sydney/Australia, November 2005. ISBN 0-7695-2421-4. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/LCN2005.pdf, doi:10.1109/LCN.2005.98. 178, 179, 187

[DR07] Thomas Dreibholz and Erwin Paul Rathgeb. On Improving the Performance of Reliable Server Pooling Systems for Distance-Sensitive Distributed Applications.

In *Proceedings of the 15. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, Informatik aktuell, pages 39–50, Bern/Switzerland, February 2007. Springer. ISBN 978-3-540-69962-0. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/KiVS2007.pdf, doi:10.1007/978-3-540-69962-0_4. 63, 178

[DR08a] Thomas Dreibholz and Erwin Paul Rathgeb. A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations. In *Proceedings of the 1st ACM/ICST International Workshop on OMNeT++*, Marseille/France, March 2008. ISBN 978-963-9799-20-2. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/OMNeTWorkshop2008.pdf, doi:10.4108/ICST.SIMUTOOLS2008.2990. 179, 181

[DR08b] Thomas Dreibholz and Erwin Paul Rathgeb. An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems. *SERSC International Journal on Hybrid Information Technology (IJHIT)*, 1(2):17–32, April 2008. ISSN 1738-9968. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/IJHIT2008.pdf. 65, 178, 187

[DR08c] Thomas Dreibholz and Erwin Paul Rathgeb. Towards the Future Internet – An Overview of Challenges and Solutions in Research and Standardization. In *Proceedings of the 2nd GI/ITG KuVS Workshop on the Future Internet*, Karlsruhe/Germany, November 2008. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/FutureInternet2008.pdf. 1, 4, 15, 65

[DR09] Thomas Dreibholz and Erwin Paul Rathgeb. Overview and Evaluation of the Server Redundancy and Session Failover Mechanisms in the Reliable Server Pooling Framework. *International Journal on Advances in Internet Technology (IJAIT)*, 2(1):1–14, June 2009. ISSN 1942-2652. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/IJAIT2009.pdf. 1, 14, 65, 177, 179, 181, 187

[DRB+07] Rudra Dutta, George N. Rouskas, Ilia Baldine, Arnold Bragg, and Dan Stevenson. The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet. In *Proceedings of the IEEE International Conference on Communications*, pages 1899–1904, Glasgow/United Kingdom, June 2007. ISBN 1-4244-0353-7. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.3035&rep=rep1&type=pdf, doi:10.1109/ICC.2007.316. 15

[Dre01] Thomas Dreibholz. Management of Layered Variable Bitrate Multimedia Streams over DiffServ with Apriori Knowledge. Masters thesis, University of Bonn, Institute for Computer Science, February 2001. Available from: http://www.iem.uni-due.de/~dreibh/diplom/Thesis.pdf. 19, 24, 92

[Dre02] Thomas Dreibholz. An Efficient Approach for State Sharing in Server Pools. In *Proceedings of the 27th IEEE Local Computer Networks Conference (LCN)*, pages 348–349, Tampa, Florida/U.S.A., November 2002. ISBN 0-7695-1591-6. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/StateSharing-Paper-ShortVersion.pdf, doi:10.1109/LCN.2002.1181806. 179

[Dre05] Thomas Dreibholz. Das rsplib–Projekt – Hochverfügbarkeit mit Reliable Server Pooling. In *Proceedings of the LinuxTag*, Karlsruhe/Germany, June 2005. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/LinuxTag2005.pdf. 186

[Dre07] Thomas Dreibholz. *Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture*. PhD thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, March 2007. Available from: http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-16326/Dre2006-final.pdf. 14, 46, 65, 78, 177, 178, 179, 186

[Dre11] Thomas Dreibholz. Server-Redundanz und Lastverteilung einfach in eigene Anwendungen integrieren – mit Reliable Server Pooling und RSPLIB. In *Proceedings of the LinuxTag*, Berlin/Germany, May 2011. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/LinuxTag2011.pdf. 65, 177, 179, 186, 188

[Dre12a] Thomas Dreibholz. An IPv4 Flowlabel Option. Internet Draft Version 15, IETF, Individual Submission, January 2012. draft-dreibholz-ipv4-flowlabel-15.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-ipv4-flowlabel-15.txt. 24

[Dre12b] Thomas Dreibholz. Applicability of Reliable Server Pooling for Real-Time Distributed Computing. Internet Draft Version 12, IETF, Individual Submission, January 2012. draft-dreibholz-rserpool-applic-distcomp-12.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-rserpool-applic-distcomp-12.txt. 179

[Dre12c] Thomas Dreibholz. Handle Resolution Option for ASAP. Internet Draft Version 10, IETF, Individual Submission, January 2012. draft-dreibholz-rserpool-asap-hropt-10.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-rserpool-asap-hropt-10.txt. 178

[DRS+11] Thomas Dreibholz, Irene Rüngeler, Robin Seggelmann, Michael Tüxen, Erwin Paul Rathgeb, and Randall R. Stewart. Stream Control Transmission Protocol: Past, Current, and Future Standardization Activities. *IEEE Communications Magazine*, 49(4):82–88, April 2011. ISSN 0163-6804. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/CommMag2011.pdf, doi:10.1109/MCOM.2011.5741151. 4, 51, 61, 63, 65, 79, 89, 104, 153, 174, 181

[DRZ08] Thomas Dreibholz, Erwin Paul Rathgeb, and Xing Zhou. On Robustness and Countermeasures of Reliable Server Pooling Systems against Denial of Service Attacks. In *Proceedings of the 7th International IFIP Networking Conference*, volume 4982 of *Lecture Notes in Computer Science*, pages 586–598, Singapore, May 2008. Springer. ISBN 978-3-540-79548-3. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/Networking2008.pdf, doi:10.1007/978-3-540-79549-0_51. 181

[DSB12] Thomas Dreibholz, Robin Seggelmann, and Martin Becke. Sender Queue Info Option for the SCTP Socket API. Internet Draft Version 03, IETF, Network Working Group,

March 2012. draft-dreibholz-tsvwg-sctpsocket-sqinfo-03.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-tsvwg-sctpsocket-sqinfo-03.txt. 145, 174

[DSTR10] Thomas Dreibholz, Robin Seggelmann, Michael Tüxen, and Erwin Paul Rathgeb. Transmission Scheduling Optimizations for Concurrent Multipath Transfer. In *Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, volume 8, Lancaster, Pennsylvania/U.S.A., November 2010. ISSN 2074-5168. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/PFLDNeT2010.pdf. 144, 173, 181

[DSV00] Thomas Dreibholz, Jan Selzer, and Simon Vey. Echtzeit-Audioübertragung mit QoS-Management in einem DiffServ-Szenario. Projektseminararbeit, Universität Bonn, Institut für Informatik, August 2000. Available from: http://www.iem.uni-due.de/~dreibh/rn/Bericht.pdf. 24

[DT03] Thomas Dreibholz and Michael Tüxen. High Availability using Reliable Server Pooling. In *Proceedings of the Linux Conference Australia (LCA)*, Perth/Australia, January 2003. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/RSerPool-Paper.pdf. 186

[DT08] Thomas Dreibholz and Michael Tüxen. Reliable Server Pooling Policies. RFC 5356, IETF, September 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5356.txt. 65, 178

[DV06] Petar Djukić and Shahrokh Valaee. Reliable Packet Transmissions in Multipath Routed Wireless Networks. *IEEE Transactions on Mobile Computing*, 5(5):548–559, May 2006. ISSN 1536-1233. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.5646&rep=rep1&type=pdf, doi:10.1109/TMC.2006.72. 68

[DWPW07] Yu Dong, Dingding Wang, Niki Pissinou, and Jian Wang. Multi-Path Load Balancing in Transport Layer. In *Proceedings of the 3rd IEEE EuroNGI Conference on Next Generation Internet Networks*, pages 135–142, Trondheim/Norway, May 2007. ISBN 1-4244-0857-1. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4231831, doi:10.1109/NGI.2007.371208. 1

[DZ12a] Thomas Dreibholz and Xing Zhou. Definition of a Delay Measurement Infrastructure and Delay-Sensitive Least-Used Policy for Reliable Server Pooling. Internet Draft Version 09, IETF, Individual Submission, January 2012. draft-dreibholz-rserpool-delay-09.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-rserpool-delay-09.txt. 178

[DZ12b] Thomas Dreibholz and Xing Zhou. Takeover Suggestion Flag for the ENRP Handle Update Message. Internet Draft Version 07, IETF, Individual Submission, January 2012. draft-dreibholz-rserpool-enrp-takeover-07.txt, work in progress. Available from: http://tools.ietf.org/id/draft-dreibholz-rserpool-enrp-takeover-07.txt. 178

[DZB+10] Thomas Dreibholz, Xing Zhou, Martin Becke, Jobin Pulinthanath, Erwin Paul Rathgeb, and Wencai Du. On the Security of Reliable Server Pooling Systems. *International Journal on Intelligent Information and Database Systems (IJIIDS)*, 4(6):552–578, December 2010. ISSN 1751-5858. Available from: http://www.tdr.wiwi.uni-due.

de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/IJIIDS2010.pdf, doi:10.1504/IJIIDS.2010.036894. 63, 179, 181

[DZR09] Thomas Dreibholz, Xing Zhou, and Erwin Paul Rathgeb. SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations. In *Proceedings of the 2nd ACM/ICST International Workshop on OMNeT++*, pages 1–8, Rome/Italy, March 2009. ISBN 978-963-9799-45-5. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/OMNeT__Workshop2009.pdf, doi:10.4108/ICST.SIMUTOOLS2009.5517. 88, 171, 181, 189, 192

[DZT06] Peter Dimopoulos, Panlop Zeephongsekul, and Zahir Tari. Multipath Aware TCP (MATCP). In *Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC)*, pages 981–988, Pula-Cagliari, Sardinia/Italy, June 2006. ISBN 0-7695-2588-1. Available from: http://researchbank.rmit.edu.au/eserv/rmit:1475/n2006000053.pdf, doi:10.1109/ISCC.2006.105. 68

[EAN+11] Nasif Ekiz, Paul D. Amer, Preethi Natarajan, Randall R. Stewart, and Janardhan R. Iyengar. Non-Renegable Selective Acknowledgements (NR-SACKs) for SCTP. Internet Draft Version 08, IETF, Network Working Group, August 2011. draft-natarajan-tsvwg-sctp-nrsack-08, work in progress. Available from: http://tools.ietf.org/id/draft-natarajan-tsvwg-sctp-nrsack-08.txt. 61

[Eat11] John W. Eaton. *GNU Octave*, 2011. Available from: http://www.gnu.org/software/octave/doc/interpreter/. 190

[EB96] Robert Elz and Randy Bush. Serial Number Arithmetic. Informational RFC 1982, IETF, August 1996. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1982.txt. 28

[Edd07] Wesley M. Eddy. TCP SYN Flooding Attacks and Common Mitigations. Informational RFC 4987, IETF, August 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4987.txt. 53

[EF94] Kjeld Borch Egevang and Paul Francis. The IP Network Address Translator (NAT). Informational RFC 1631, IETF, May 1994. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1631.txt. 74

[EJ01] Donald E. Eastlake and Paul E. Jones. US Secure Hash Algorithm 1 (SHA1). Informational RFC 3174, IETF, September 2001. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3174.txt. 184, 186

[FGM+99] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Standards Track RFC 2616, IETF, June 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2616.txt. 49

[Fis00] Eric Fischer. The Evolution of Character Codes, 1874-1968, 2000. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.678&rep=rep1&type=pdf. 14

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993. ISSN 1063-6692. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.128.5092&rep=rep1&type=pdf, doi:10.1109/90.251892. 23, 105

[FJQ+08] Sheila Fallon, Paul Jacob, Yuansong Qiao, Liam Murphy, Enda Fallon, and Austin Hanley. SCTP Switchover Performance Issues in WLAN Environments. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC 2008)*, pages 564–568, January 2008. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.7424&rep=rep1&type=pdf, doi:10.1109/CCNC08.2007.131. 63

[FKNT02] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Grid Service Infrastructure WG, Global Grid Forum*, June 2002. Available from: http://www.globus.org/alliance/publications/papers/ogsa.pdf. 178

[Flo97] Sally Floyd. RED: Discussions of Setting Parameters, November 1997. Available from: http://icir.org/floyd/REDparameters.txt. 23, 105, 161

[For88] Chuck Forsberg. *The ZMODEM Inter Application File Transfer Protocol*, October 1988. Available from: http://gallium.inria.fr/~doligez/zmodem/zmodem.txt. 18

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. *GRID Today*, July 2002. Available from: http://dlib.cs.odu.edu/WhatIsTheGrid.pdf. 178

[Fre11] FreeBSD Documentation Project. *FreeBSD Handbook*, 2011. Available from: ftp://ftp.freebsd.org/pub/FreeBSD/doc/en/books/handbook/book.pdf.bz2. 90

[FRH+11] Alan Ford, Costin Raiciu, Mark Handley, Sébastien Barré, and Janardhan R. Iyengar. Architectural Guidelines for Multipath TCP Development. Informational RFC 6182, IETF, March 2011. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6128.txt. 2, 74

[FRHB11] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. Internet Draft Version 04, IETF, Individual Submission, July 2011. draft-ietf-mptcp-multiaddressed-04, work in progress. Available from: http://tools.ietf.org/id/draft-ietf-mptcp-multiaddressed-04.txt. 74

[FV10] Kevin Fall and Kannan Varadhan. *The NS Manual*, May 2010. Available from: http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf. 77

[Get11a] Jim Gettys. Bufferbloat – Dark Buffers in the Internet, January 2011. Available from: http://www.bufferbloat.net/attachments/9/BufferBloat11.pdf. 116, 161

[Get11b] Jim Gettys. What is Bufferbloat, Anyway?, 2011. Available from: http://gettys.wordpress.com/what-is-bufferbloat-anyway/. 116, 161

[GKT00] Klaus D. Gradischnig, Stefan Krämer, and Michael Tüxen. Loadsharing – A Key to the Reliability of SS7-Networks. In *Proceedings of the Second International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 216–221, Munich/Germany, April 2000. ISBN 3896759280. Available from: http://www.sctp.de/papers/drcn2000.pdf. 151

[Got10] Aitor Goti. *Discrete Event Simulations*. Sciyo, Rijeka/Croatia, August 2010. ISBN 978-953-307-115-2. Available from: http://www.intechopen.com/books/show/title/discrete-event-simulations. 78

[Gro02] Dan Grossman. New Terminology and Clarifications for DiffServ. Informational RFC 3260, IETF, April 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3260.txt. 24

[HAN02] Thomas J. Hacker, Brian D. Athey, and Brian Noble. The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, Fort Lauderdale, Florida/U.S.A., 2002. ISBN 0-7695-1573-8. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.22.5958&rep=rep1&type=pdf. 69

[HBWW99] Juha Heinanen, Fred Baker, Walter Weiss, and John Wroclawski. Assured Forwarding PHB Group. Standards Track RFC 2597, IETF, June 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2597.txt. 24

[HBZ07] Jogi Hofmüller, Aaron Bachmann, and Iohannes Zmoelnig. The Transmission of IP Datagrams over the Semaphore Flag Signaling System (SFSS). Informational RFC 4824, IETF, April 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4824.txt. 47

[HDU12] Carsten Hohendorf, Thomas Dreibholz, and Esbold Unurkhaan. Secure SCTP. Internet Draft Version 13, IETF, Individual Submission, January 2012. draft-hohendorf-secure-sctp-13.txt, work in progress. Available from: http://tools.ietf.org/id/draft-hohendorf-secure-sctp-13.txt. 62

[Hem05] Stephen Hemminger. Network Emulation with NetEm. In *Proceedings of the Linux Conference Australia (LCA)*, Canberra/Australia, April 2005. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.1687&rep=rep1&type=pdf. 100

[HG02] Ibrahim F. Haddad and David Gordon. Network Simulator 2: A Simulation Tool for Linux. *Linux Journal*, October 2002. Available from: http://www.linuxjournal.com/node/5929/print. 77

[HH06] Paul Hoffman and Susan Harris. The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force. Informational RFC 4677, IETF, September 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4677.txt. 43

[Hic11] Ian Hickson. HTML5 – A Vocabulary and Associated APIs for HTML and XHTML. Working draft, W3C, April 2011. work in progress. Available from: http://www.w3.org/TR/2011/WD-html5-20110405/Overview.html. 49

[Hop00] Christian E. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. Informational RFC 2992, IETF, November 2000. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2992.txt. 68

[HŘS+06] Petr Hlávka, Vojtěch Řehák, Aleš Smrčka, David Šafránek, Pavel Šimeček, and Tomáš Vojnar. Formal Verification of the CRC Algorithm Properties. In *Second Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, pages 55–62, 2006. ISBN 80-214-3287-X. Available from: http://www.fit.vutbr.cz/~smrcka/pub/fmcrc-MEMICS06.pdf. 27

[HRUT07] Carsten Hohendorf, Erwin Paul Rathgeb, Esbold Unurkhaan, and Michael Tüxen. Secure End-to-End Transport Over SCTP. *Journal of Computers*, 2(4):31–40, June 2007. ISSN 1796-203X. Available from: http://www.academypublisher.com/jcp/vol02/no04/jcp02043140.html. 62

[HS02] Hung-Yun Hsieh and Raghupathy Sivakumar. pTCP: An End-to-End Transport Layer Protocol for Striped Connections. In *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*, pages 24–33, Paris/France, November 2002. ISBN 0-7695-1856-7. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.396&rep=rep1&type=pdf, doi:10.1109/ICNP.2002.1181383. 68, 70

[IAS05] Janardhan R. Iyengar, Paul Amer, and Randall Stewart. Receive Buffer Blocking in Concurrent Multipath Transfer. In *Proceedings of the IEEE GLOBECOM*, pages 121–126, St. Louis, Missouri/U.S.A., November 2005. ISBN 978-1-4244-1707-0. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.7496&rep=rep1&type=pdf, doi:10.1109/GLOCOM.2005.1577365. 112

[IAS06] Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Concurrent Multipath Transfer using SCTP Multihoming over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, October 2006. ISSN 1063-6692. Available from: http://www.fandm.edu/jiyengar/papers/cmt-ton2006.pdf, doi:10.1109/TNET.2006.882843. 2, 70, 81, 89, 90, 103, 104, 105, 126, 151, 172, 174

[ITU93] ITU-T. Introduction to CCITT Signalling System No. 7. Recommendation Q.700, International Telecommunication Union, March 1993. Available from: http://www.item.ntnu.no/fag/ttm4130/stottelitteratur/T-REC-Q.700.pdf. 65

[ITU94] ITU-T. Open Systems Interconnection – Base Reference Model. Recommentation X.200, International Telecommunication Union, August 1994. Available from: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items. 11, 13

[JB88] Van Jacobson and Robert Braden. TCP Extensions for Long-Delay Paths. Standards Track RFC 1072, IETF, October 1988. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1072.txt. 34

[JBB92] Van Jacobson, Robert Braden, and David A. Borman. TCP Extensions for High Performance. Informational RFC 1323, IETF, May 1992. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1323.txt. 69

[JDS06] JDSU. *ADSL Technology – Overview, Line Qualification and Service Turn-up*. Vienna/Austria, April 2006. Available from: http://www.jdsu.com/ProductLiterature/ADSL_Technology_White_Paper.pdf. 42

[JK88] Van Jacobson and Michael J. Karels. Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, 18:314–329, August 1988. ISSN 0146-4833. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.3262&rep=rep1&type=pdf, doi:10.1145/52325.52356. 37, 41

[JNM+04] Petri Jokela, Pekka Nikander, Jan Melen, Jukka Ylitalo, and Jorma Wall. Host Identity Protocol. In *Proceedings of the Wireless World Research Forum (WWRF)*, Beijing/People's Republic of China, February 2004. Available from: http://www.jokela.org/publications/wwrf8bis.pdf. 75

[JR06] Andreas Jungmaier and Erwin Paul Rathgeb. On SCTP Multi-Homing Performance. *Telecommunication Systems*, 31(2-3):141–161, March 2006. ISSN 1018-4864. Available from: http://www.springerlink.com/content/m444211644h30510/fulltext.pdf, doi:10.1007/s11235-006-6517-7. 2

[JRT02] Andreas Jungmaier, E. P Rathgeb, and Michael Tüxen. On the Use of SCTP in Failover-Scenarios. In *Proceedings of the State Coverage Initiatives, Mobile/Wireless Computing and Communication Systems II*, volume X, pages 363–368, Orlando, Florida/U.S.A., July 2002. ISBN 980-07-8150-1. Available from: http://tdrwww.iem.uni-due.de/inhalt/forschung/sctp_fb/sctp-failover.pdf. 1, 63, 151

[Jun05] Andreas Jungmaier. *Das Transportprotokoll SCTP*. PhD thesis, Universität Duisburg-Essen, Institut für Experimentelle Mathematik, August 2005. Available from: http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-13244/dissertation_jungmaier.pdf. 2, 65, 77

[KHF06] Eddie Kohler, Mark Handley, and Sally Floyd. Datagram Congestion Control Protocol (DCCP). Standards Track RFC 4340, IETF, March 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4340.txt. 48, 54, 92, 96

[Koo02] Philip Koopman. 32-Bit Cyclic Redundancy Codes for Internet Applications. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 459–472, Washington, DC/U.S.A., June 2002. ISBN 0-7695-1597-5. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.8323&rep=rep1&type=pdf. 27

[KP91] Phil Karn and Craig Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems (TOCS)*, 9:364–373, November 1991. ISSN 0734-2071. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.122.7350&rep=rep1&type=pdf, doi:10.1145/55482.55484. 41

[KR08] James Kurose and Keith Ross. *Computernetzwerke: Der Top-Down-Ansatz*. Pearson Studium, 2008. ISBN 978-3827373304. 23, 42, 43, 48, 49, 50, 69, 85

[Kuz05] Aleksandar Kuzmanović. The Power of Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review*, 35:61–72, October 2005. ISSN 0146-4833. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.793&rep=rep1&type=pdf, doi:10.1145/1090191.1080100. 41

[Lah00] Kevin Lahey. TCP Problems with Path MTU Discovery. Informational RFC 2923, IETF, September 2000. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2923.txt. 48

[LOTD08] Peter Lei, Lyndon Ong, Michael Tüxen, and Thomas Dreibholz. An Overview of Reliable Server Pooling Protocols. Informational RFC 5351, IETF, September 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5351.txt. 1, 65, 177

[LSW12] Ulf Lamping, Richard Sharpe, and Ed Warnicke. Wireshark User's Guide, February 2012. Available from: http://www.wireshark.org/download/docs/user-guide-a4.pdf. 96

[MA01] Matt Mathis and Mark Allman. A Framework for Defining Empirical Bulk Transfer Capacity Metrics. Informational RFC 3148, IETF, July 2001. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3148.txt. 19, 103

[Mal93] Gary Scott Malkin. Traceroute Using an IP Option. Standards Track RFC 1393, IETF, January 1993. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1393.txt. 69

[Mal98] Gary Scott Malkin. RIP Version 2. Standards Track RFC 2453, IETF, November 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2453.txt. 9, 49

[Max75] Nicholas F. Maxemchuk. Dispersity Routing. In *Proceedings of the IEEE International Conference on Communications (ICC)*, volume 41, pages 10–13, San Francisco, California/U.S.A., June 1975. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.2508&rep=rep1&type=pdf. 67

[Max07] Nicholas F. Maxemchuk. Dispersity Routing: Past and Present. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 1–7, Orlando, Florida/U.S.A., October 2007. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4455301, doi:10.1109/MILCOM.2007.4455301. 67

[MBB+97] Allison Mankin, Fred Baker, Bob Braden, Scott Bradner, Michael O'Dell, Allyn Romanow, A. Weinrib, and Lixia Zhang. Resource ReSerVation Protocol (RSVP) – Version 1 Applicability Statement – Some Guidelines on Deployment. Informational RFC 2208, IETF, September 1997. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2208.txt. 24

[MCR+06] Al Morton, Len Ciavattone, Gomathi Ramachandran, Stanislav Shalunov, and Jerry Perser. Packet Reordering Metrics. Standards Track RFC 4737, IETF, November 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4737.txt. 23

[Mes05] James Messer. comp.dcom.lans.ethernet Frequently Asked Questions, January 2005. Available from: http://www.NetworkUptime.com/faqs/ethernet. 22

[MH07] Matt Mathis and John W. Heffner. Packetization Layer Path MTU Discovery. Standards Track RFC 4821, IETF, March 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4821.txt. 48

[MK01] Luiz Magalhaes and Robin Kravets. Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts. In *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP)*, pages 165–171, Riverside, California/U.S.A., November 2001. ISBN 0-7695-1429-4. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.8201&rep=rep1&type=pdf, doi:10.1109/ICNP.2001.992896. 68

[MLE+99] Louis Mamakos, Kurt Lidl, Jeff Evarts, David Carrel, Dan Simone, and Ross Wheeler. A Method for Transmitting PPP Over Ethernet (PPPoE). Informational RFC 2516, IETF, February 1999. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2516.txt. 47

[MM97] Gary Scott Malkin and Robert E. Minnear. RIPng for IPv6. Standards Track RFC 2080, IETF, January 1997. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2080.txt. 50

[MMBK10] David L. Mills, Jim Martin, Jack Burbank, and William Kasch. Network Time Protocol Version 4: Protocol and Algorithms. Standards Track RFC 5905, IETF, June 2010. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5905.txt. 49

[MMFR96] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgment Options. Standards Track RFC 2018, IETF, October 1996. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2018.txt. 48

[MN06] Robert Moskowitz and Pekka Nikander. Host Identity Protocol (HIP) Architecture. Informational RFC 4423, IETF, May 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4423.txt. 75

[MNJH08] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas R. Henderson. Host Identity Protocol. RFC 5201, IETF, April 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5201.txt. 75

[Moy98] John T. Moy. OSPF Version 2. Standards Track RFC 2328, IETF, April 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2328.txt. 50

[MP93] Gary Scott Malkin and Tracy LaQuey Parker. Internet Users' Glossary. Informational RFC 1392, IETF, January 1993. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1392.txt. 11, 18

[MR06] A. Maharana and G. N. Rathna. Fault-tolerant Video on Demand in RSerPool Architecture. In *Proceedings of the International Conference on Advanced Computing and Communications (ADCOM)*, pages 534–539, Bangalore/India, December 2006. ISBN 1-4244-0716-8. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4289950, doi:10.1109/ADCOM.2006.4289950. 179

[MZ08] Robert Morelos-Zaragoza. The Error Correcting Codes (ECC) Page, August 2008. Available from: http://www.eccpage.com/. 28

[NA08] Thomas Narten and Harald Tveit Alvestrand. Guidelines for Writing an IANA Considerations Section in RFCs. Informational RFC 5226, IETF, May 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5226.txt. 46

[Nag84] John Nagle. Congestion Control in IP/TCP Internetworks. Informational RFC 896, IETF, January 1984. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc896.txt. 35, 37, 39

[NB09] Erik Nordmark and Marcelo Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. Standards Track RFC 5533, IETF, June 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5533.txt. 75

[NBBB98] Kathleen Nichols, Steven Blake, Fred Baker, and David L. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Technical Report 2474, IETF, December 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2474.txt. 24

[NEA+08] Preethi Natarajan, Nasif Ekiz, Paul D. Amer, Janardhan R. Iyengar, and Randall Stewart. Concurrent Multipath Transfer using SCTP Multihoming: Introducing the Potentially-Failed Destination State. In *Proceedings of the 7th International IFIP Networking Conference*, pages 727–734, Singapore, May 2008. Springer. ISBN 978-3-540-79548-3. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.5495&rep=rep1&type=pdf. 63

[NEY+08] Preethi Natarajan, Nasif Ekiz, Ertuğrul Yilmaz, Paul D. Amer, and Janardhan R. Iyengar. Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP. In *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP)*, pages 187–196, Orlando, Florida/U.S.A., October 2008. ISBN 978-1-4244-2506-8. Available from: http://www.cis.udel.edu/~amer/PEL/poc/pdf/ICNP2008-natarajanNonRenegableSacks.pdf, doi:10.1109/ICNP.2008.4697037. 62, 128, 172, 174

[NIS07] NIST. The NIST Reference on Constants, Units and Uncertainty – Prefixes for Binary Multiples, March 2007. Available from: http://physics.nist.gov/cuu/Units/binary.html. 20

[NN11] Yoshifumi Nishida and Preethi Natarajan. Quick Failover Algorithm in SCTP. Internet Draft Version 04, IETF, Network Working Group, September 2011. draft-nishida-tsvwg-sctp-failover-04, work in progress. Available from: http://tools.ietf.org/id/draft-nishida-tsvwg-sctp-failover-04.txt. 63

[PA00] Vern Paxson and Mark Allman. Computing TCP's Retransmission Timer. Standards Track RFC 2988, IETF, November 2000. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2988.txt. 39, 41

[PAMM98] Vern Paxson, Guy Almes, Jamshid Mahdavi, and Matt Mathis. Framework for IP Performance Metrics. Informational RFC 2330, IETF, May 1998. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2330.txt. 19

[Pos80] Jonathan Bruce Postel. User Datagram Protocol. Standards Track RFC 768, IETF, August 1980. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc768.txt. 48, 92

[Pos81a] Jonathan Bruce Postel. Internet Control Message Protocol. Standards Track RFC 792, IETF, September 1981. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc792.txt. 48

[Pos81b] Jonathan Bruce Postel. Internet Protocol. Standards Track RFC 791, IETF, September 1981. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc791.txt. 18, 24, 47

[Pos81c] Jonathan Bruce Postel. Transmission Control Protocol. Standards Track RFC 793, IETF, September 1981. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc793.txt. 17, 48, 53, 54, 74, 92

[PR85] Jonathan Bruce Postel and J. Reynolds. File Transfer Protocol (FTP). Standards Track RFC 959, IETF, October 1985. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc959.txt. 15, 49

[PTIW07] Brad Penoff, Mike Tsai, Janardhan R. Iyengar, and Alan Wagner. Using CMT in SCTP-based MPI to Exploit Multiple Interfaces in Cluster Nodes. In *Proceedings of the EuroPVM/MPI*, Paris/France, September 2007. ISBN 978-3-540-75415-2. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.4547&rep=rep1&type=pdf, doi:10.1007/978-3-540-75416-9_31. 85, 103, 107

[QBC+08] Jürgen Quittek, Stewart Bryant, Benoît Claise, Paul Aitken, and Jeff Meyer. Information Model for IP Flow Information Export. Standards Track RFC 5102, IETF, January 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5102.txt. 65

[R D11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. Vienna/Austria, July 2011. Available from: http://cran.r-project.org/doc/manuals/refman.pdf. 181, 190

[RBP+11] Costin Raiciu, Sébastien Barré, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proceedings of the ACM SIGCOMM*, Toronto/Canada, August 2011. Available from: http://inl.info.ucl.ac.be/system/files/mptcp-sigcomm.pdf. 75

[RFB01] K. K. Ramakrishnan, Sally Floyd, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. Standards Track RFC 3168, IETF, September 2001. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3168.txt. 41

[RHW09] Costin Raiciu, Mark Handley, and Damon Wischik. Practical Congestion Control for Multipath Transport Protocols. Technical report, University College London, London/United Kingdom, 2009. Available from: http://nrg.cs.ucl.ac.uk/mptcp/mptcp-techreport.pdf. 153, 154, 155, 173

[RHW11] Costin Raiciu, Mark Handley, and Damon Wischik. Coupled Multipath-Aware Congestion Control. Internet Draft Version 07, IETF, Network Working Group, July 2011. draft-ietf-mptcp-congestion-07, work in progress. Available from: http://tools.ietf.org/id/draft-ietf-mptcp-congestion-07.txt. 154

[Rij94] Anil Rijsinghani. Computing the Internet Checksum via Incremental Update. Informational RFC 1624, IETF, May 1994. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1624.txt. 25

[RKT02] Dan Rubenstein, Jim Kurose, and Don Towsley. Detecting Shared Congestion of Flows via End-to-End Measurement. *IEEE/ACM Transactions on Networking*, 10(3):381–395, June 2002. ISSN 1063-6692. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.131.6332&rep=rep1&type=pdf, doi:10.1109/TNET.2002.1012369. 152

[RLH06] Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). Standards Track RFC 4271, IETF, January 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4271.txt. 50

[RMMG11] Maher Ali Al Rantisi, Ali Maqousi, Glenford Mapp, and Orhan Gemikonakli. The Development of a Dynamic and Robust Event-Based Routing Protocol in Wireless

Sensor Networks for Environmental Monitoring. In *Proceedings of the 1st IEEE Conference on Communication, Science and Information Engineering (CCSIE)*, London/United Kingdom, July 2011. Available from: http://eprints.mdx.ac.uk/8122/1/ccsie2011_submission_30-1.pdf. 68

[Ros06] Timothy Roscoe. The End of Internet Architecture. In *Proceedings of 5th ACM Workshop on Hot Topics in Networks (HotNets-V)*, Irvine, California/U.S.A., November 2006. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2772&rep=rep1&type=pdf. 15

[RRAW98] Antony Richards, Glynn Rogers, Mark Antoniades, and Varuni Witana. Mapping User Level QoS from a Single Parameter. In *Proceedings of the 2nd International Conference on Multimedia Networks and Services (MMNS)*, Versailles/France, 1998. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.5059&rep=rep1&type=pdf. 19

[RSB+05] Thibault Renier, Hans-Peter Schwefel, Marjan Božinovski, Kim Larsen, Ramjee Prasad, and Robert Seidl. Distributed Redundancy or Cluster Solution? An Experimental Evaluation of Two Approaches for Dependable Mobile Internet Services. *Lecture Notes in Computer Science*, 3335, January 2005. ISBN 978-3-540-24420-2. Available from: http://www.springerlink.com/content/pe4wlbrxd4kkla2e/fulltext.pdf. 179

[RTR08] Irene Rüngeler, Michael Tüxen, and Erwin Paul Rathgeb. Integration of SCTP in the OMNeT++ Simulation Environment. In *Proceedings of the 1st ACM/ICST International Workshop on OMNeT++*, Marseille/France, March 2008. ISBN 978-963-9799-20-2. Available from: http://portal.acm.org/citation.cfm?id=1416310, doi:10.4108/ICST.SIMUTOOLS2008.3027. 77, 79, 80

[RTR09] Irene Rüngeler, Michael Tüxen, and Erwin Paul Rathgeb. Considerations on Handling Link Errors in SCTP. *ICB Research Reports, University of Duisburg-Essen*, 35, August 2009. ISSN 1860-2770. Available from: http://www.icb.uni-due.de/fileadmin/ICB/research/research_reports/ICBReportNo35.pdf. 63, 129

[Rün09] Irene Rüngeler. *SCTP – Evaluating, Improving and Extending the Protocol for Broader Deployment*. PhD thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, December 2009. Available from: http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-23465/DissPDF.pdf. 62, 77

[RW97] Erwin Paul Rathgeb and Eugen Wallmeier. *ATM – Infrastruktur für die Hochleistungskommunikation*. Springer-Verlag, Berlin/Germany, September 1997. ISBN 978-3540603702. 37, 42

[SA11a] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Address Format. Standards track rfc, IETF, March 2011. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6122.txt. 49

[SA11b] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Standards Track RFC 6120, IETF, March 2011. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6120.txt. 49

[SA11c] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Standards Track RFC 6121, IETF, March 2011. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6121.txt. 49

[Sav06] Pekka Savola. IPv6 Site Multihoming Using a Host-based Shim Layer. In *Proceedings of the 5th IEEE International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL)*, April 2006. ISBN 0-7695-2552-0. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.1246&rep=rep1&type=pdf, doi:10.1109/ICNICONSMCL.2006.131. 75

[SBCQ09] Ganesh Sadasivan, Nevil Brownlee, Benoît Claise, and Jürgen Quittek. Architecture for IP Flow Information Export. Informational RFC 5470, IETF, March 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5470.txt. 65

[SBG00] Harimath Sivakumar, Stuart Bailey, and Robert L. Grossman. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, Dallas, Texas/U.S.A., November 2000. ISSN 1063-9535. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.6140&rep=rep1&type=pdf, doi:10.1109/SC.2000.10040. 69

[SC05] Pekka Savola and Tim Chown. A Survey of IPv6 Site Multihoming Proposals. In *Proceedings of the 8th IEEE International Conference on Telecommunications (ConTEL)*, volume 1, pages 41–48, Zagreb/Croatia, June 2005. ISBN 953-184-081-4. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.2213&rep=rep1&type=pdf, doi:10.1109/CONTEL.2005.185815. 76

[SCFJ03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. Standards Track RFC 3550, IETF, July 2003. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3550.txt. 23

[SCH+99] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas E. Anderson. The End-to-End Effects of Internet Path Selection. In *Proceedings of the ACM SIGCOMM*, pages 289–299, Cambridge, Massachusetts/U.S.A., August 1999. ISBN 1-58113-135-6. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.38.9273&rep=rep1&type=pdf, doi:10.1145/316194.316233. 68

[Sch10] Michael Scharf. Multi-Connection TCP (MCTCP) Transport. Internet Draft Version 01, IETF, Individual Submission, July 2010. draft-scharf-mptcp-mctcp-01, work in progress. Available from: http://tools.ietf.org/id/draft-scharf-mptcp-mctcp-01.txt. 69

[SDR08] Pascal Schöttle, Thomas Dreibholz, and Erwin Paul Rathgeb. On the Application of Anomaly Detection in Reliable Server Pooling Systems for Improved Robustness against Denial of Service Attacks. In *Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 207–214, Montréal, Québec/Canada, October 2008. ISBN 978-1-4244-2413-9. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/LCN2008.pdf, doi:10.1109/LCN.2008.4664171. 181

[SE01] Pyda Srisuresh and Kjeld Borch Egevang. Traditional IP Network Address Translator (Traditional NAT). Informational RFC 3022, IETF, January 2001. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3022.txt. 74

[Sew07] Julian Seward. *bzip2 – A Program and Library for Data Compression*, December 2007. Available from: http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.pdf. 95, 184

[Sim94] William Allen Simpson. The Point-to-Point Protocol (PPP). Standards Track RFC 1661, IETF, July 1994. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1661.txt. 47

[SLT12] Randall R. Stewart, Peter Lei, and Michael Tüxen. Stream Control Transmission Protocol (SCTP) Packet Drop Reporting. Internet Draft Version 13, IETF, Individual Submission, February 2012. draft-stewart-sctp-pktdrprep-13.txt, work in progress. Available from: http://tools.ietf.org/id/draft-stewart-sctp-pktdrprep-13.txt. 62

[SM05] Vineet Srivastava and Mehul Motani. Cross-Layer Design: A Survey and the Road Ahead. *IEEE Communications Magazine*, 43(12):112–119, December 2005. ISSN 0163-6804. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1561928, doi:10.1109/MCOM.2005.1561928. 15

[SMPB02] Greg Sidebottom, Ken Morneault, and Javier Pastor-Balbas. Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) – User Adaptation Layer (M3UA). Standards Track RFC 3332, IETF, September 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3332.txt. 145

[SMS10] Richard M. Stallman, Roland McGrath, and Paul D. Smith. *GNU Make – A Program for Directing Recompilation*, July 2010. Available from: http://www.gnu.org/software/make/manual/make.pdf. 181, 184

[Sol92] Karen R. Sollins. The TFTP Protocol (Revision 2). Standards Track RFC 1350, IETF, July 1992. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1350.txt. 49

[SP03] Tommy Svensson and Alex Popescu. OPNET Modeler – Development of Laboratory Exercises based on OPNET Modeler. Master's thesis, Blekinge Institute of Technology, Karlskrona/Sweden, June 2003. Available from: http://www.opnet.com/university_program/teaching_with_opnet/textbooks_and_materials/materials/Lab_Exercices_Modeler.pdf. 77

[SPG97] Scott Shenker, Craig Partridge, and Roch Guerin. Specification of Guaranteed Quality of Service. Standards Track RFC 2212, IETF, September 1997. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2212.txt. 24

[SRX+04] Randall R. Stewart, M. Ramalho, Qiaobing Xie, Michael Tüxen, and Phillip T. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. Standards Track RFC 3758, IETF, May 2004. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3758.txt. 61

[SSO02] Jonathan Stone, Randall R. Stewart, and Douglas Otis. Stream Control Transmission Protocol (SCTP) Checksum Change. Standards Track RFC 3309, IETF, September 2002. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3309.txt. 26, 27, 52

[Ste00] Ralf Steinmetz. *Multimedia-Technologie – Grundlagen, Komponenten und Systeme*. Springer, Berlin/Germany, 3rd edition, August 2000. ISBN 978-3540673323. 17, 19, 24, 30, 42, 49, 100

[Ste07] Randall R. Stewart. Stream Control Transmission Protocol. Standards Track RFC 4960, IETF, September 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4960.txt. 1, 27, 49, 51, 52, 54, 58, 59, 60, 82, 92, 104, 105, 126, 140, 141, 142, 144, 151, 155, 158, 161

[STL12] Randall R. Stewart, Michael Tüxen, and Peter Lei. Stream Control Transmission Protocol (SCTP) Stream Reconfiguration. Standards track rfc, IETF, February 2012. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6525.txt. 61

[STP+11] Randall R. Stewart, Michael Tüxen, Kacheong Poon, Peter Lei, and Vladislav Yasevich. Sockets API Extensions for Stream Control Transmission Protocol (SCTP). Informational rfc, IETF, December 2011. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc6458.txt. 63, 89, 145

[STR10] Robin Seggelmann, Michael Tüxen, and Erwin Paul Rathgeb. Stream Scheduling Considerations for SCTP. In *Proceedings of the 18th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, September 2010. ISBN 978-953-290-004-0. Available from: http://ieeexplore.ieee.org/iel5/5611454/5623609/05623661.pdf?arnumber=5623661. 144

[SX01] Randall R. Stewart and Qiaobing Xie. *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison-Wesley, Amsterdam/Holland, 2001. ISBN 0-201721-86-4. 51, 63

[SXM+00] Randall R. Stewart, Qiaobing Xie, Ken Morneault, Chip Sharp, Hanns Jürgen Schwarzbauer, Tom Taylor, Ian Rytina, Malleswar Kalla, Lixia Zhang, and Vern Paxson. Stream Control Transmission Protocol. Standards Track RFC 2960, IETF, October 2000. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2960.txt. 51

[SXST08a] Randall R. Stewart, Qiaobing Xie, Maureen Stillman, and Michael Tüxen. Aggregate Server Access Protcol (ASAP). RFC 5352, IETF, September 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5352.txt. 49, 65, 178

[SXST08b] Randall R. Stewart, Qiaobing Xie, Maureen Stillman, and Michael Tüxen. Aggregate Server Access Protocol (ASAP) and Endpoint Handlespace Redundancy Protocol (ENRP) Parameters. RFC 5354, IETF, September 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5354.txt. 65

[SXT+07] Randall R. Stewart, Qiaobing Xie, Michael Tüxen, Shin Maruyama, and Masahiro Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. Standards Track RFC 5061, IETF, September 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5061.txt. 61

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, Upper Saddle River, New Jersey/U.S.A., 1996. ISBN 0-13-349945-6. 13, 20, 27, 28, 34, 36, 42, 43, 48, 49, 50, 67, 79, 101

[TB08] Brian H. Trammell and Elisa Boschi. Bidirectional Flow Export Using IP Flow Information Export. Standards Track RFC 5103, IETF, January 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5103.txt. 65

[TGFS+09] Phuoc Tran-Gia, Anja Feldmann, Ralf Steinmetz, Jörg Eberspächer, Martina Zitterbart, Paul Müller, and Hans Schotten. *G-Lab White Paper Phase 1 – Studien und Experimentalplattform für das Internet der Zukunft*, January 2009. Available from: https://www.german-lab.de/fileadmin/Press/G-Lab_White_Paper_Phase1.pdf. 102

[TH00] Dave Thaler and Christian E. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. Informational RFC 2991, IETF, November 2000. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2991.txt. 68

[TH01] Aristotelis Tsirigos and Zygmunt J. Haas. Multipath Routing in the Presence of Frequent Topological Changes. *IEEE Communications Magazine*, 39(11):132–138, November 2001. ISSN 0163-6804. Available from: http://www.ee.oulu.fi/~carlos/papers/routing/TS01.pdf, doi:10.1109/35.965371. 68

[TRR08] Michael Tüxen, Irene Rüngeler, and Erwin Paul Rathgeb. Interface Connecting the INET Simulation Framework with the Real World. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, pages 1–6, Marseille/France, March 2008. ISBN 978-963-9799-20-2. Available from: http://dl.acm.org/citation.cfm?doid=1416222.1416267, doi:10.1145/1416222.1416267. 79

[TRS11] Michael Tüxen, Irene Rüngeler, and Randall R. Stewart. SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol. Internet Draft Version 08, IETF, Individual Submission, October 2011. draft-tuexen-tsvwg-sctp-sack-immediately-08.txt, work in progress. Available from: http://tools.ietf.org/id/draft-tuexen-tsvwg-sctp-sack-immediately-08.txt. 62

[TS12] Michael Tüxen and Randall R. Stewart. UDP Encapsulation of SCTP Packets. Internet Draft Version 03, IETF, Transport Area Working Group, March 2012. draft-ietf-tsvwg-sctp-udp-encaps-03.txt, work in progress. Available from: http://tools.ietf.org/id/draft-ietf-tsvwg-sctp-udp-encaps-03.txt. 63

[TSLR07] Michael Tüxen, Randall R. Stewart, Peter Lei, and Eric Rescorla. Authenticated Chunks for the Stream Control Transmission Protocol (SCTP). Standards Track RFC 4895, IETF, August 2007. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4895.txt. 61

[Uij09] Henk Uijterwaal. A One-Way Packet Duplication Metric. Standards Track RFC 5560, IETF, May 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5560.txt. 23

[Unu05] Esbold Unurkhaan. *Secure End-to-End Transport – A New Security Extension for SCTP*. PhD thesis, University of Duisburg-Essen, Institute for Experimental Mathematics, July 2005. Available from: http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-13053/Thesis%20Esbold%20Unurkhaan.pdf. 62

[URJ04] Esbold Unurkhaan, Erwin Paul Rathgeb, and Andreas Jungmaier. Secure SCTP – A Versatile Secure Transport Protocol. *Telecommunication Systems*, 27(2-4):273–296, 2004. ISSN 1018-4864. Available from: http://www.springerlink.com/content/l8mph67087w17107/fulltext.pdf. 62

[UZF+03] Ümit Uyar, Jianliang Zheng, Mariusz A. Fecko, Sunil Samtani, and Phillip T. Conrad. Reliable Server Pooling for Future Combat Systems. In *Proceedings of the IEEE MILCOM Military Communications Conference*, volume 2, pages 927–932, Boston, Massachusetts/U.S.A., October 2003. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.9480&rep=rep1&type=pdf. 179

[UZF+04] Ümit Uyar, Jianliang Zheng, Mariusz A. Fecko, Sunil Samtani, and Phillip T. Conrad. Evaluation of Architectures for Reliable Server Pooling in Wired and Wireless Environments. *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks*, 22(1):164–175, 2004. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.2471&rep=rep1&type=pdf, doi:10.1109/JSAC.2003.818806. 179

[Var10] András Varga. *OMNeT++ Discrete Event Simulation System User Manual – Version 4.1*, December 2010. Available from: http://www.omnetpp.org/doc/omnetpp41/Manual.pdf. 77, 78, 85, 186

[Var12] András Varga. INET Framework for OMNeT++/OMNEST release 20111118-0cc8077, 2012. Available from: http://inet.omnetpp.org/doc/INET/neddoc/index.html. 78, 80

[Vyn09] Eric Vyncke. IPv6 over Social Networks. RFC 5514, IETF, April 2009. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5514.txt. 47

[Wai90] David Waitzman. Standard for the Transmission of IP Datagrams on Avian Carriers. RFC 1149, IETF, April 1990. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc1149.txt. 47

[Wel05] Michael Welzl. *Network Congestion Control: Managing Internet Traffic*. John Wiley & Sons, Chichester, West Sussex/United Kingdom, 2005. ISBN 978-0-470-02528-4. 37, 41

[WHB08] Damon Wischik, Mark Handley, and Marcelo Bagnulo Braun. The Resource Pooling Principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, October 2008. ISSN 0146-4833. Available from: http://ccr.sigcomm.org/online/files/p47-handleyA4.pdf, doi:10.1145/1452335.1452342. 3, 152, 173

[Wro97] John Wroclawski. Specification of the Controlled-Load Network Element Service. Standards Track RFC 2211, IETF, September 1997. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc2211.txt. 24

[XSS+08] Qiaobing Xie, Randall R. Stewart, Maureen Stillman, Michael Tüxen, and Aron J. Silverton. Endpoint Handlespace Redundancy Protocol (ENRP). RFC 5353, IETF, September 2008. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc5353.txt. 46, 65, 178

[Yer03] François Yergeau. UTF-8, A Transformation Format of ISO 10646. Standards Track RFC 3629, IETF, November 2003. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc3629.txt. 14

[YL06] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Connection Protocol. Standards Track RFC 4254, IETF, January 2006. ISSN 2070-1721. Available from: http://www.ietf.org/rfc/rfc4254.txt. 49, 145

[YWY08a] Muhammad Murtaza Yousaf, Michael Welzl, and Bülent Yener. Accurate Shared Bottleneck Detection Based On SVD and Outliers Detection. Technical Report NSG-DPS-UIBK-01, University of Innsbruck, Institute of Computer Science, Innsbruck/Austria, August 2008. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.3147&rep=rep1&type=pdf. 152, 161

[YWY08b] Muhammad Murtaza Yousaf, Michael Welzl, and Bülent Yener. On the Accurate Identification of Network Paths having a Common Bottleneck. In *Proceedings of the ACM SIGCOMM*, Seattle, Washington/U.S.A., August 2008. Poster Presentation. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.1874&rep=rep1&type=pdf. 152, 161

[ZDB+10] Xing Zhou, Thomas Dreibholz, Martin Becke, Jobin Pulinthanath, Erwin Paul Rathgeb, and Wencai Du. The Software Modeling and Implementation of Reliable Server Pooling and RSPLIB. In *Proceedings of the 8th ACIS Conference on Software Engineering Research, Management and Applications (SERA)*, pages 129–136, Montréal, Québec/Canada, May 2010. ISBN 978-0-7695-4075-7. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/SERA2010.pdf, doi:10.1109/SERA.2010.26. 179

[ZDR07] Xing Zhou, Thomas Dreibholz, and Erwin Paul Rathgeb. A New Approach of Performance Improvement for Server Selection in Reliable Server Pooling Systems. In *Proceedings of the 15th IEEE International Conference on Advanced Computing and Communication (ADCOM)*, pages 117–121, Guwahati/India, December 2007. ISBN 0-7695-3059-1. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/ReliableServer/Publications/ADCOM2007.pdf, doi:10.1109/ADCOM.2007.19. 187

[ZDR08] Wenyu Zhu, Thomas Dreibholz, and Erwin Paul Rathgeb. Analysis and Evaluation of a Scalable QoS Device for Broadband Access to Multimedia Services. In *Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN)*, pages 504–505, Montréal, Québec/Canada, October 2008. ISBN 978-1-4244-2413-9. Available from: http://www.tdr.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/FlowRouting/Paper/LCN2008-FlowRouting.pdf, doi:10.1109/LCN.2008.4664212. 181

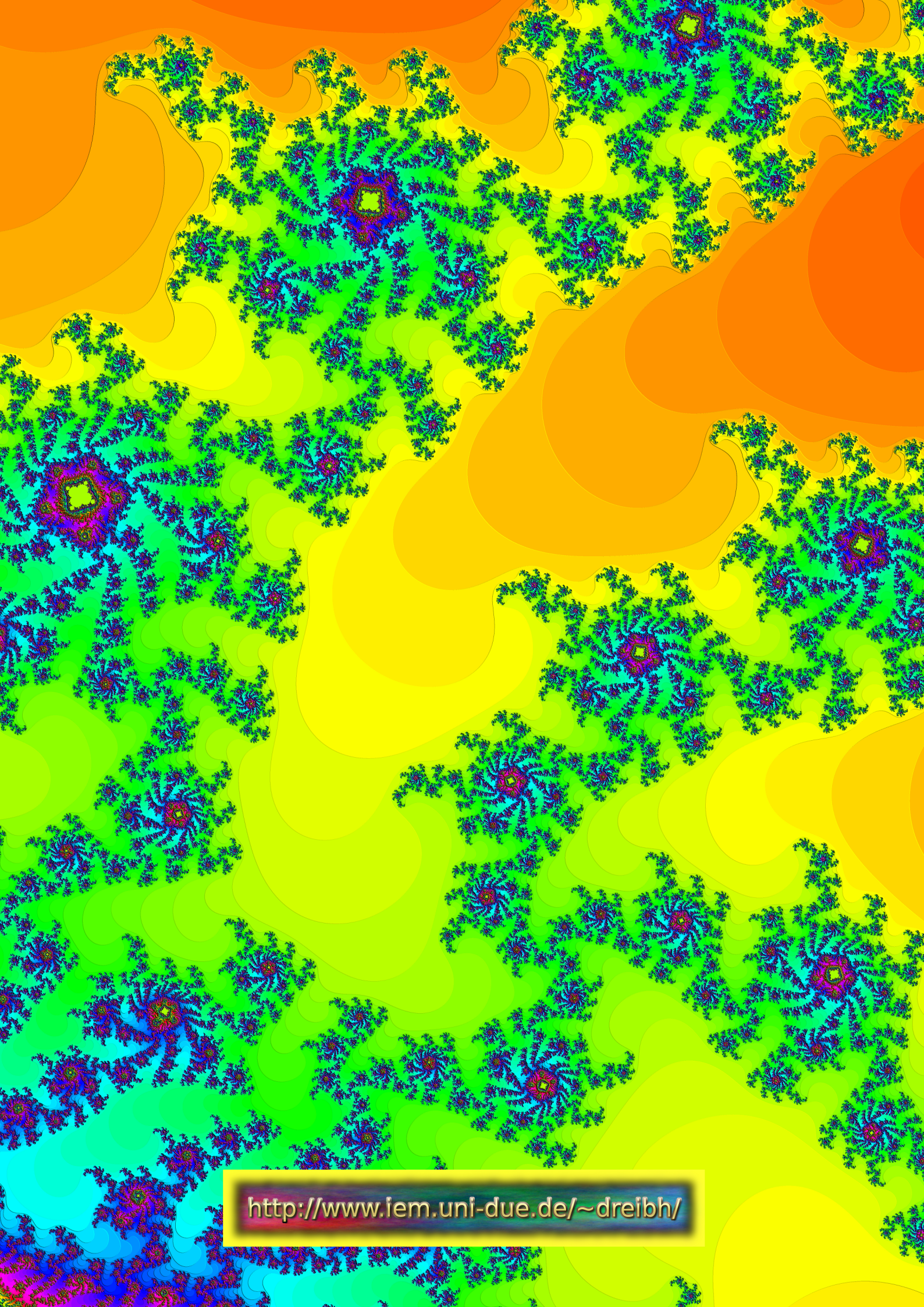
[ZDRZ09] Wenyu Zhu, Thomas Dreibholz, Erwin Paul Rathgeb, and Xing Zhou. A Scalable QoS Device for Broadband Access to Multimedia Services. *SERSC International Journal of Multimedia and Ubiquitous Engineering (IJMUE)*, 4(2):157–172, May 2009. ISSN 1975-0080. Available from: http://www.sersc.org/journals/IJMUE/vol4_no2_2009/14.pdf. 181

[ZLK+04] Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry Peterson, and Randolph Wang. A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths. In *In Proceedings of the USENIX Annual Technical Conference*, pages 99–112, Boston, Massachusetts/U.S.A., June 2004. Available from: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.111.3109&rep=rep1&type=pdf. 69

Index

Curriculum Vitae

Name:	Thomas Dreibholz
29.09.1976	born in Bergneustadt, Nordrhein-Westfalen, Germany
08/1983 - 09/1987	Student at the Grundschule Wiehl-Bielstein (Primary School), Germany
09/1987 - 06/1993	Student at the Realschule Wiehl-Bielstein (Junior High School), Germany http://www.realschule-wiehl.de
08/1993 - 07/1996	Student at the Gymnasium Wiehl (High School), Germany http://www.dbgwiehl.de
08/1996 - 04/2001	Student of Computer Science at the University of Bonn, Germany http://www.uni-bonn.de
since 10/1998	Vordiplom (Bachelor's Degree) of Computer Science at the University of Bonn, Germany
since 04/2001	Diplom (Master's Degree) of Computer Science at the University of Bonn, Germany
since 03/2007	Ph.D. (Dr. rer. nat.) of Computer Science at the University of Duisburg-Essen, Germany
since 05/2001	Assistant Professor in the Computer Networking Technology Group at the Institute for Experimental Mathematics of the University of Duisburg-Essen in Essen, Germany http://tdrwww.iem.uni-due.de
since 2002	Cisco™ Certified Network Associate (CCNA) and Cisco™ Certified Academy Instructor (CCAI) at the Cisco™ Networking Academy of the University of Duisburg-Essen in Essen, Germany http://cna.uni-due.de


http://www.iem.uni-due.de/~dreibh/