Rate Adaptation for Hierarchical Packet Schedulers Considering Traffic Differentiation by Congestion Control

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde
eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

David P. Wagner

geb. in Bonn-Bad Godesberg

Hauptberichter:	Prof. Dr.-Ing. Andreas Kirstädter
Mitberichter:	Prof. Dr. Michael Menth
Tag der mündlichen Prüfung:	12. März 2019

Institut für Kommunikationsnetze und Rechnersysteme
der Universität Stuttgart

2019

To Thea and my parents.

Summary

In today's wired broadband Internet access networks, the aggregation links often constitute bottlenecks at times of peak load. This results in Quality of Experience (QoE) deterioration during these periods. The downstream Best Effort (BE) traffic in such networks contains a significant amount of deferrable traffic, which does not contribute to immediate QoE. A large share of this background traffic behaves differently from foreground traffic because it uses special background Congestion Control Algorithms (CCAs), e.g. BitTorrent's uTorrent Transport Protocol (uTP). When competing with a standard foreground CCA for a bottleneck's bandwidth, such background traffic yields. The use of background and foreground CCAs builds a system for traffic differentiation based on an implicit signaling, that is, the bottleneck's queuing delay. For most downstream traffic in the regional access network, the bottleneck is located in this access network, i.e. in the hierarchical packet scheduler of its Broadband Network Gateway (BNG).

Today's Internet Service Providers (ISPs) either accept performance degradation during peak periods or apply traffic management systems. These traffic management systems are based on bandwidth consumption or on traffic classification, typically supported by Deep Packet Inspection (DPI). In the research community, the concepts of Congestion Policing based on Congestion Exposure and Congestion Policing Queue (CPQ) have been proposed and evaluated. These approaches have substantial drawbacks since they either ignore the senders' implicit priority signaling or deployment in regional access networks is not feasible.

This thesis aims to improve QoE during peak periods by adapting the BNG's hierarchical packet scheduling.

The core concept of Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload (RADICCO) is identifying foreground and background traffic and adapting the scheduling weights during overload. RADICCO operates on access link level, i.e. on per-subscriber level. It recognizes foreground and background traffic by observing the size of respective queues in the hierarchical scheduler. Based on the recognized traffic types, it increases the bandwidth of the foreground traffic and reduces the bandwidth of background traffic. This thesis presents an algorithm implementing the two functionalities, discusses design decisions and alternatives, and evaluates the proposed concept.

The performance of the presented algorithm is evaluated by simulations that incorporate the unmodified CCA implementations of the Linux kernel as well as BitTorrent's libutp. In these simulations, RADICCO's impact on QoE-relevant Quality of Service (QoS) is measured for four

traffic models. The results show that RADICCO improves QoE for all modeled services if the BNG is the bottleneck.

The contribution of this thesis is threefold: First, this work combines two fields in packet switched networks, namely Medium Access Control (MAC) layer packet scheduling and transport layer Congestion Control (CC). Although the two inevitably interact, the interaction is often neglected in the respective research. This thesis proposes a novel mechanism that exploits understanding transport layer CC to derive an adaptation of the packet scheduling, which considers its impact on the traffic's CC. Second, this thesis defines an algorithm that implements this concept for hierarchical schedulers with an arbitrary number of hierarchy levels. The algorithm consists of a traffic type recognition function and a rate adaptation function for peak periods. Third, RADICCO is evaluated using a prototype implementation and simulations incorporating CCA implementations widely used in today's Internet. The evaluation proves the effectiveness of RADICCO.

Kurzfassung

In heutigen drahtgebundenen Breitbandzugangsnetzen bilden zu Spitzenlastzeiten oft Aggregationsverbindungen Engpässe. Dies führt zu einer Verschlechterung des Nutzererlebnisses. In diesen Netzen enthält der nicht priorisierte Verkehr in Richtung der Kundenanschlüsse einen signifikanten Anteil von aufschiebbaren Übertragungen, die nicht zum unmittelbaren Nutzererlebnis beitragen. Ein großer Teil dieses Hintergrundverkehrs verhält sich anders als der Vordergrundverkehr, weil er besondere Überlastregelungssalgorithmen (ÜRAs) verwendet, z.B. das uTorrent Transport Protocol (uTP) von BitTorrent. Wenn ein solcher Hintergrund-ÜRA mit einem Standard Vordergrund-ÜRA an einem Engpass um Bandbreite konkurriert, gibt der Hintergrund-ÜRA nach. Das Verwenden von Hintergrund- und Vordergrund-ÜRAs bildet ein System für Verkehrsdifferenzierung, das auf impliziter Signalisierung basiert, nämlich der Wartenschlangenverzögerung am Engpass. Für den meisten Verkehr in Richtung der Kundenanschlüsse in einem regionalen Zugangsnetz befindet sich der Engpass in diesem Zugangsnetz, d.h. im hierarchischen Scheduler des Netzabschlussrouters.

Heutige Internetprovider akzeptieren entweder die Leistungseinschränkung in Spitzenlastzeiten oder setzen Systeme zur Verkehrsverwaltung ein. Diese Systeme basieren entweder auf der Bandbreitennutzung oder auf Verkehrsklassifizierung, typischerweise unterstützt durch detaillierte Paketanalyse (Deep Packet Inspection). In der Forschungsgemeinde wurden die Konzepte „Überlastüberwachung basierend auf dem Preisgeben von Überlast“ (Congestion Policing based on Congestion Exposure) und „Überlast überwachende Warteschlange“ (Congestion Policing Queue) vorgeschlagen und untersucht. Diese Konzepte haben erhebliche Nachteile, da sie entweder die implizite Priorisierungssignalisierung des Senders ignorieren oder ein praktischer Einsatz im regionalen Zugangsnetz nicht möglich ist.

Das in dieser Dissertation verfolgte Ziel ist es, das Nutzererlebnis während Spitzenlastzeiten durch das Anpassen des hierarchischen Schedulings des Netzabschlussrouters zu verbessern.

Das Kernkonzept von „Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload“ (RADICCO) ist es, Vordergrund- und Hintergrundverkehr zu erkennen und während Spitzenlastzeiten die Gewichte des Schedulers anzupassen. RADICCO arbeitet auf Ebene der Internetanschlüsse, d.h. auf Kundenebene. Es erkennt Vordergrund- und Hintergrundverkehr anhand der Beobachtung der entsprechenden Warteschlangengrößen im hierarchischen Scheduler. Basierend auf den erkannten Verkehrstypen erhöht es die Bandbreite des Vordergundverkehrs und senkt die Bandbreite des Hintergrundverkehrs. Diese Dissertation stellt einen Algorithmus vor, der diese beiden Funktionalitäten implementiert, diskutiert Entwurfsentscheidungen und ihre Alternativen und untersucht das vorgeschlagene Verfahren.

Die Leistung des vorgestellten Algorithmus wird durch Simulationen untersucht, die die unveränderten ÜRA-Implementierungen des Linux-Kernels und von BitTorrents libutp integrieren. In diesen Simulationen wird die durch RADICCO bewirkte Veränderung der für das Nutzererlebnis relevanten Dienstgüte für vier Verkehrsmodelle gemessen. Die Ergebnisse zeigen, dass RADICCO bei einem Engpass im Netzabschlussrouter das Nutzererlebnis für alle abgebildeten Dienste verbessert.

Diese Dissertation leistet drei Beiträge: Erstens kombiniert diese Arbeit zwei Gebiete in paketvermittelten Netzen, nämlich das Paket Scheduling der Medienzugangskontrollschicht und die Überlastregelung der Transportschicht. Obwohl beide unweigerlich interagieren, wird diese Interaktion in der jeweiligen Forschung oft nicht beachtet. Diese Dissertation schlägt ein neuartiges Verfahren vor, das ein Verständnis von Transportschichtüberlastregelung nutzt, um eine Anpassung des Paket Scheduling abzuleiten, die ihren Einfluss auf die Überlastregelung des Verkehrs berücksichtigt. Zweitens definiert diese Dissertation einen Algorithmus, der dieses Konzept für hierarchische Scheduler mit einer beliebigen Anzahl von Hierarchieebenen implementiert. Der Algorithmus besteht aus einer Funktion zur Erkennung des Verkehrstyps und einer Funktion zur Anpassung der Raten während Spitzenlastzeiten. Drittens wird RADICCO mittels einer prototypischen Implementierung und Simulationen untersucht, die im heutigen Internet verbreitet genutzte ÜRA-Implementierungen integrieren. Die Untersuchung belegt die Wirksamkeit von RADICCO.

Contents

List of Figures

List of Tables

List of Abbreviations

IKR	Institute of Communication Networks and Computer Engineering
3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorization and Accounting
ABC	Appropriate Byte Count
ACK	Acknowledgment
ADSL	Asymmetric Digital Subscriber Line
ADSL2	ADSL 2
ADSL2+	Extended bandwidth ADSL2
AGS	Aggregation Switch
AIAD	Additive Increase, Additive Decrease
AIMD	Additive Increase, Multiplicative Decrease
ANCP	Access Node Control Protocol
AN	Access Node
AQM	Active Queue Management
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BBRR	Bit-by-Bit Round Robin
BDP	Bandwidth Delay Product
BE	Best Effort
BIC	Binary Increase Congestion Control
BITS	Background Intelligent Transfer Service
BNG	Broadband Network Gateway

BNetzA	Bundesnetzagentur
BRAS	Broadband Remote Access Server
CAPEX	Capital Expenditure
CBR	Constant Bit Rate
CC	Congestion Control
CCA	CC Algorithm
CDN	Content Delivery Network
CPC	Congestion Policing based on ConEx
CPQ	Congestion Policing Queue
CPU	Central Processing Unit
CoDel	Controlling Queue Delay
ConEx	Congestion Exposure
DASH	Dynamic Adaptive Streaming over HTTP
DCTCP	Data Center TCP
DC	Data Center
DOCSIS	Data Over Cable Service Interface Specification
DPI	Deep Packet Inspection
DRR	Deficit Round Robin
DS-Lite	Dual-Stack Lite
DSCP	Differentiated Services Code Point
DSLAM	Digital Subscriber Line Access Multiplexer
DSL	Digital Subscriber Line
DWDM	Dense Wavelength Division Multiplexing
DiffServ	Differentiated Services
ECN-CE	ECN-Congestion Encountered
ECN	Explicit Congestion Notification
EWMA	Exponentially Weighted Moving Average
FCC	Federal Communications Commission

FCFS	First-Come, First-Served
FEC	Forward Error Correction
FIFO	First In, First Out
FQ	Fair Queueing
FRR	Fair Round Robin
FTTB	Fiber To The Building
FTTC	Fiber To The Cabinet/Curb
FTTH	Fiber To The Home
FullHD	Full High Definition
G.fast	Fast Access to Subscriber Terminals, ITU G.9701
GMPLS	Generalized Multi-Protocol Label Switching
GPS	Generalized Processor Sharing
GRE	Generic Routing Encapsulation
GRO	Generic Receive Offload
H-TCP	Hamilton TCP
HFC	Hybrid Fiber Coax
HFS	Hierarchical Fair Scheduler
HTTP	Hyper Text Transport Protocol
IAT	Inter-Arrival Time
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IP	Internet Protocol
ISP	Internet Service Provider
ITU	International Telecommunication Union
IXP	Internet Exchange Point
IoT	Internet of Things

LAN	Local Area Network
LBE	Lower than Best Effort
LEDBAT	Low Extra Delay Background Transport
LFVC	Leap Forward Virtual Clock
M2M	Machine-to-Machine
MAC	Medium Access Control
MIAD	Multiplicative Increase, Additive Decrease
MIMD	Multiplicative Increase, Multiplicative Decrease
MIPS	Million Instructions Per Second
MTU	Maximum Transmission Unit
NAPT	Network Address and Port Translation
NAS	Network Access Server
NAT	Network Address Translation
NDP	Neighbor Discovery Protocol
NG-PON2	40-Gigabit-Capable PON2
OSI Model	Open Systems Interconnection Model
OS	Operating System
OWD	One-Way Delay
P2P	Peer-to-Peer
PEP	Policy Enforcement Point
PGPS	Packetized Generalized Processor Sharing
PON	Passive Optical Network
PRR	Proportional Rate Reduction
QoE	Quality of Experience
QoS	Quality of Service
RADICCO	Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload
RED	Random Early Detection
RFC	Request For Comment

RR	Round Robin
RTCP	RTP Control Protocol
RTO	Retransmission TimeOut
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
SACK	Selective Acknowledgment
SEFF	Smallest Eligible virtual Finish time First
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
TDM	Time-Division Multiplexing
TOE	TCP Offload Engine
TSO	TCP Segmentation Offload
TV	Television
UDP	User Datagram Protocol
USA	United States of America
VBR	Variable Bit Rate
VC	Virtual Circuit
VDSL	Very High Speed Digital Subscriber Line
VDSL2	VDSL 2
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network
VoD	Video on Demand
VoIP	Voice over IP
WF^{2}Q+	The successor of WF^{2}Q with lower complexity
WF^{2}Q	Worst-case Fair Weighted Fair Queueing
WFI	Worst-case Fair Index
WFQ	Weighted Fair Queueing

XG-PON	10-Gigabit-PON
cwnd	Congestion Window
mDRR	modified DRR
nWFI	normalized Worst-case Fair Index
rmcat	RTP Media Congestion Avoidance Techniques
ssthresh	slow start threshold
uTP	uTorrent Transport Protocol

List of Symbols

C_R	The capacity of the scheduler's root node R, i.e. of the Broadband Network Gateway (BNG)'s downstream link.
C_e	The capacity of edge node e, i.e. of the downstream link from the aggregation switch to Access Node (AN) e.
$C_{i,j}$	The capacity of leaf node i,j, i.e. of the downstream link of the j's subscriber attached to AN i, subscriber i,j.
$w_i^{TT}(t)$	The target rate for the traffic type TT of inner node i at time t.
$w_{i,j}^{FG}$	The target rate of leaf node i,j if recognized as carrying foreground traffic. For all foreground leaf nodes holds at any time: $w_{i,j}^{FG} = C_{i,j}$.
$w_{i,j}^{BG}(t)$	the target rate of leaf node i,j at time t if recognized as carrying background traffic at time t.
$r_{i,j}^{FG}(t)$	The effective rate of leaf node i,j at time t if recognized as carrying foreground traffic at time t.
$r_{i,j}^{BG}(t)$	The effective rate of leaf node i,j at time t if recognized as carrying background traffic at that time t.
$sw_i^{FG}(t)$	The sum of target rates for foreground traffic of all child nodes of inner node i.
$sw_i^{BG}(t)$	The sum of target rates for background traffic of all child nodes of inner node i.
$o_i(t)$	The local load factor of inner node i.
$w_i^{FG}(t)$	The target rate of foreground traffic at inner node i.
$w_i^{BG}(t)$	The target rate of background traffic at inner node i.

$r_{MAX_i}^{BG}(t)$ The maximum possible rate of background traffic at inner node i due to restrictions of links transporting the background traffic further down the hierarchy.

$sr_{MAX_i}^{BG}(t)$ The sum of maximum possible rates of background traffic of all child nodes at inner node i.

$\bar{h}_i^{BG}(t)$ The best-case headroom for background traffic at inner node i, calculated based on the capacity C_i.

$h_i^{BG}(t)$ The headroom for background traffic at inner node i, calculated based on the effective rate $r_i(t)$.

$f_i(t)$ The fill up fraction at inner node i at time t that defines the fraction of best-case headroom for background traffic that is actually allocated to each child of i.

$r_i^{FG}(t)$ The effective rate of foreground traffic at inner node i.

$r_i^{BG}(t)$ The effective rate of background traffic at inner node i.

$r_i(t)$ The effective rate of inner node i. Calculated as $r_i(t) = r_i^{FG}(t) + r_i^{BG}(t)$.

1 Introduction

Today, wired broadband Internet access plays an important role in most people's life in industrial countries. It is used for a large and growing set of service types, ranging from home automation to Video on Demand (VoD) streaming. Since traffic is mostly triggered by human activity, the load of access networks follows a diurnal pattern. In combination with economical link dimensioning this creates a problem during peak periods: At these times, the Internet access link of many subscribers does not deliver its contracted bandwidth, i.e. bitrate, as aggregation links in the regional access network become bottlenecks during these periods, deteriorating the Quality of Experience (QoE). Yet, not all transported traffic is of equal importance or urgency: While some traffic such as VoD streaming contributes to QoE since it results in an immediate user experience, other traffic has no short-term impact on QoE, e.g. file sharing or software update downloads. Moreover, not only QoE-relevant, i.e. foreground traffic, but also a substantial share of so-called background traffic is transported during periods of peak load. Therefore, QoE deterioration due to overload of aggregation interfaces could be avoided, if the background transfers were deferred.

Nevertheless, up to now there is no satisfactory solution available. Some Internet Service Providers (ISPs) try to identify foreground and background traffic by carrying out Deep Packet Inspection (DPI) and then assign different priorities to the identified services. However, this approach has several drawbacks: DPI-based service identification is error-prone and works inadequately in case of encrypted traffic. Moreover, these systems base on the ISP's rating of services only and provide no incentive for senders to cooperate, e.g. to communicate openly. Introducing and obeying explicit priority signaling, e.g. based on Differentiated Services (DiffServ), would avoid uncertainty and false classification, but is no feasible alternative for three reasons: First, the traffic concerned originates from external sources. Thus, an in-band signaling is not possible due to the lack of a trust relationship between the traffic sources and the delivering ISP. Second, such an approach would require changes to many devices and configurations, and is therefore likely to cause undesired deterioration of services not implementing the new signaling. Lastly, such system would require an incentive not to mark all traffic as high priority. With today's flat rate Internet plans, such incentive could only exist in the technical domain, but that fundamentally contradicts the core concept of implementing priorities according to the subscribers' requests. So, DiffServ or other explicit priority signaling is not used in today's access networks.

An analysis of the situation in wired access networks shows that a signaling for priority differentiation exists and is already in use. Moreover, it leaves the decision on the priority to the receiver and provides a robust incentive system. This system for traffic differentiation is based on the end hosts using different transport layer CC Algorithms (CCAs) that result in unbalanced resource sharing at bottlenecks, i.e. the different CCAs correspond to different priorities. Priority

differentiation by Congestion Control (CC) works if the bottleneck is shared, that means a shared queue enables indirect feedback on the bottleneck's congestion to all connections. Today, Apple's software updates and BitTorrent's Peer-to-Peer (P2P) service use a CCA designed as background CCA, and are responsible for a significant traffic share.

A subscriber's access link is an element of the lowest level of a hierarchical topology, which is rooted at a Broadband Network Gateway (BNG)'s downstream interface. This topology typically features two or three levels with decreasing link speeds from BNG to access link. In capacity dimensioning, the ISPs use oversubscription, i.e. the bandwidth of an aggregation link is lower than the sum of capacities of all links fed by it. In contrast, the ISP's core or metro network that feeds the BNG usually is contention-free. Therefore, by design, there is never a bottleneck in this part of the network. So, for most traffic destined to broadband subscribers the bottleneck is located in the regional access network downstream of the BNG, during normal load at the access link and during peak load often at an aggregation link.

To enforce Quality of Service (QoS) guarantees and to avoid deployment of such functionality to every node in the regional access network, BNGs implement hierarchical scheduling at their downstream interfaces. In the hierarchical packet scheduler, every link in the hierarchical topology is represented by a scheduling node. It ensures that packets for this link are only transmitted on the BNG's interface if and when they can be forwarded by the respective link. Typically, access networks use three or four traffic classes to support QoS guarantees for certain traffic, e.g. the ISP's Voice over IP (VoIP) and VoD services. In that case, the scheduling tree has one level more than the network topology. This hierarchical scheduling causes any congestion in the regional access network to occur not at the limiting link, but within the hierarchical packet scheduler. On each hierarchy level, today's schedulers deployed at BNG downstream interfaces share the bandwidth available for Best Effort (BE) traffic among all active child nodes proportionally fairly. This isolates the traffic of each subscriber from effects caused by traffic of other subscriber and thus prevents CC-based traffic differentiation from functioning across subscribers.

Summarizing, a solution to the described problem must combine the advantages of a hierarchical scheduler in terms of QoS enforcement and of considering traffic differentiation by CCA in terms of QoE optimization. In order not to lose the guarantees of the hierarchical scheduler, the solution proposed in this thesis is based on a hierarchical scheduler but adapts its weights. Such solution must consist of two functional components:

1. A mechanism for recognizing the traffic type, i.e. for distinguishing foreground from background traffic.
2. A means to adjust the resource allocation.

Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload (RADICCO), proposed in this thesis, recognizes the traffic type based on buffer utilization and distinguishes two types of traffic: foreground traffic and background traffic. During overload, RADICCO adapts the weights of the hierarchical scheduler to adjust resource allocation to the priorities signaled by the end hosts to optimize the overall QoE. This weight adaptation interacts with the senders' CC as well as with the traffic type detection, which is considered in the algorithm design.

Consequently, RADICCO transports more of the more important foreground traffic, and less important background transfers are deferred when bandwidth is scarce. This results in advantages for both the ISPs and the users, i.e. the subscribers. The ISP benefits from a smoother utilization of his infrastructure, a higher average utilization and, most importantly, from a deferred and reduced pressure for investments in infrastructure. For subscribers, the most important benefit is increased QoE during peak periods compared to the same situation operated without RADICCO. This benefit is made possible by automatically postponing non-urgent traffic of all subscribers during overload, so it is a mutual benefit.

1.1 Contributions

The three main contributions of this thesis are:

1. A novel concept called RADICCO for improving the overall performance by combining packet scheduling with traffic differentiation by CC.
2. An algorithm elaborating this concept.
3. An evaluation of this algorithm that uses real-word CCA implementations.

The relevance of each contribution is outlined in the following.

This dissertation brings together packet scheduling and transport layer CC in one single mechanism. Among the mechanisms of the Medium Access Control (MAC) layer, according to the Open Systems Interconnection Model (OSI Model), some Active Queue Managements (AQMs) aim to impact Transmission Control Protocol (TCP) CCAs. The literature on QoS enforcing mechanisms however, in particular on packet schedulers, usually neglects interconnections to higher layers. Likewise, transport layer CCAs are designed to perceive the network as black box and only selectively use information the network provides. Yet, these two mechanisms inevitably interact. This thesis proposes an extension to packet schedulers that exploits knowledge on transport layer CCAs to adapt the packet scheduling so that a better overall performance in terms of QoE is achieved. To the author's knowledge, it is the first packet scheduling mechanism that explicitly considers the implicit signaling of end hosts selecting different CCAs. Furthermore, this mechanism takes into account the impact of these changes on the transport layer control loops.

This thesis also defines an algorithm that implements this concept for hierarchical schedulers with an arbitrary number of hierarchy levels. It shows that the approach can be implemented with computational effort that allows implementation in today's devices and thus deployment. The algorithm consists of two functional blocks: Firstly, a traffic type recognition function and secondly, a rate adaptation function. The traffic type recognition function identifies foreground and background traffic by its buffer utilization. During peak load periods, the rate adaptation function reduces the rates of subscribers receiving only background traffic and re-distributes bandwidth among aggregation links so that subscribers receiving foreground traffic are allocated more capacity than their proportionally fair share.

Finally, a prototype implementation is used to evaluate the performance of the algorithm when facing realistic traffic. The simulations use unmodified transport layer CCA implementations

that are widely-used in today's Internet. The simulative evaluation is based on simulations with four traffic models, two background CCAs, two access network topologies and a wide range of shares of background traffic. During simulations, various statistical data on the traffic as well as on the internal states of RADICCO are gathered.

1.2 Outline

This thesis is organized in five chapters, including this introduction in Chapter 1.

Chapter 2 gives an overview on the relevant background for this thesis. Since RADICCO targets a specific problem in today's networks, and moreover combines two fields of communication networks, this chapter covers a broad range: Section 2.1 presents today's access networks for broadband Internet, outlining their role in the ISP network, the impact of access link technologies on access network topologies and typical traffic patterns in such networks. Section 2.2 covers general packet scheduling and the properties crucial for BNG schedulers. Section 2.3 introduces the topic of congestion and its relevance for and prevalence in today's Internet. Section 2.4 focuses on transport layer CC. It explains the system of differentiating traffic by using different CCAs and the relation of CC to buffer sizing.

Chapter 3 is focused on the novel approach called RADICCO. First, the motivation of this thesis is explained in Section 3.1 and the technical problem statement is outlined in Section 3.2. Then, Section 3.3 presents the core concept of RADICCO. Next, the qualitative design objectives and quantitative performance objectives are defined in Section 3.4. Related work that aims for similar goals is presented in Section 3.5. Before presenting RADICCO, Section 3.6 details the assumptions and prerequisites RADICCO is based on and discusses their future validity. The following Section 3.7 presents the algorithm of RADICCO and introduces the calculations for internal states and the effectively allocated rates. Finally, Section 3.8 discusses core design decisions and potential alternatives.

Chapter 4 provides an evaluation of RADICCO, including a performance evaluation based on simulations. It starts with an evaluation of the qualitative objectives in Section 4.1. Section 4.2 presents our approach of performance evaluation. It details the integration of wide-spread CCA implementations, the used topologies, our approach to scale overload, the four traffic models used in our simulations and our parameterizations. Furthermore, Section 4.3 presents the performance metrics, some of which correspond directly to defined objectives, e.g. bottleneck utilization. For QoE, a primary objective, we estimate if RADICCO achieves a QoE improvement by measuring QoS parameters that, for the modeled service type, decide on QoE. Moreover, the reliability of RADICCO's traffic type recognition function is evaluated. Sections 4.4, 4.5, 4.6 and 4.7 present the results for the four traffic models and Section 4.8 summarizes the evaluation.

Chapter 5 draws conclusions focusing on the benefits for the affected players and outlines potential changes a deployment of RADICCO-enabled BNGs could foster. Lastly, this section outlines fields for further research and classifies the respective research questions.

2 Background

This chapter provides the background necessary for RADICCO. Since RADICCO exploits today's network architecture and topologies, Section 2.1 presents today's access networks for broadband Internet, outlining their role in the ISP network, the impact of access link technologies on access network topologies and typical traffic patterns in such networks. Section 2.2 presents general packet scheduling and the properties crucial for BNG schedulers, the schedulers that RADICCO is designed to extend. In Section 2.4, the congestion phenomenon and the mechanisms for congestion control are introduced. This section also details how traffic differentiation is implemented by different CCAs and the relation of CC to buffer sizing.

2.1 Wired Broadband Internet Access

In this section, three topics are presented: First, the role of regional access networks and in particular of the BNG in the overall ISP network is outlined in Section 2.1.1. Second, the internal topologies and their relation to the used access technology are detailed in Section 2.1.2. Lastly, Section 2.1.3 provides information on the typical traffic patterns in access networks and typical link dimensioning.

2.1.1 Topologies and Architectures of Internet Service Provider Networks

Today's ISP networks are divided into several domains with clearly separated functions and different technologies: The core network, the metro or aggregation networks, and the regional access networks or short access network, which we focus on. Figure 2.1 shows a schematic illustration of a typical ISP network architecture. The core network of an ISP provides reliable long-distance high-bandwidth connectivity, typically between few core nodes located in larger cities. Therefore, it works on large traffic aggregates and uses high capacity Dense Wavelength Division Multiplexing (DWDM) fiber-optical networking. At the core nodes, metro or aggregation networks attach. Typically, today's aggregation networks are optical networks, often constructed as ring or several stages of rings, forming a hierarchical topology of links of decreasing speed [TR-134].

The nodes in this aggregation topology that translate links of a lower speed level to a higher speed level are called Aggregation Switch (AGS). Attached to these aggregation networks, i.e. to its lowest level, are the access networks. Usually each access network is served by one Access Node (AN), e.g. a Digital Subscriber Line Access Multiplexer (DSLAM) serving several Digital

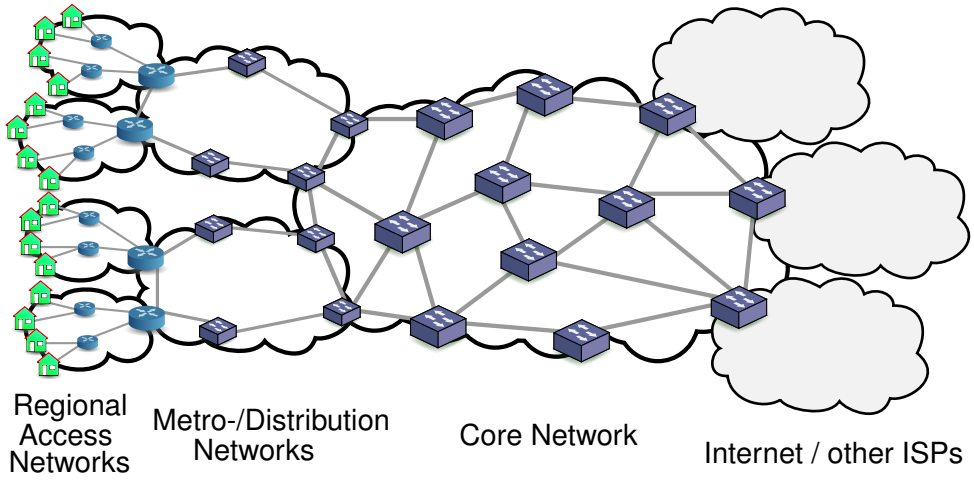


Figure 2.1: Schematic illustration of the typical high level ISP network architecture

Subscriber Line (DSL) access links. One AN may also serve several access networks. In that case, it usually serves one access network per downstream interface using a shared-medium technology. While the links in aggregation networks are usually symmetric, i.e. downstream capacity equals upstream capacity, access links are asymmetric for most technologies and in particular for the widely-deployed ones such as DSL. The access link connects the AN to the subscriber's handover point, often called home gateway. While the network beyond the home gateways is under control of the respective subscribers, the home gateway itself usually receives at least fundamental configuration from the ISP, e.g. which frequency bands to use for transmission on the upstream channel. In many networks, it even is owned and controlled by the ISP, so it is part of the ISP's network. While the control on the home gateway makes a difference with respect to upstream transmissions, it does not matter for downstream. For downstream, the access link is the last transmission controlled by the ISP, so the ANs are the layer two edge nodes according to the OSI Model.

The core and aggregation networks carry different types of traffic with a wide range of applicable QoS guarantees: They transport BE traffic, typically from and to residential subscribers. They also carry traffic with specific QoS guarantees regarding delay, loss, bandwidth or/and availability. A significant share of that traffic receives soft guarantees compared to the other QoS criteria because it belongs to private customers. Nevertheless, it is prioritized compared to BE traffic because some services should work reliably, e.g. VoIP and provider-prioritized video streaming services. Then there often is also a third group of traffic with tighter QoS guarantees in all aspects, such as traffic of business customers. Obviously, in such a tiered hierarchical network architecture the number of devices increases exponentially with decreasing aggregation level: Typically, there are just a handful of routers at core nodes but thousands of ANs.

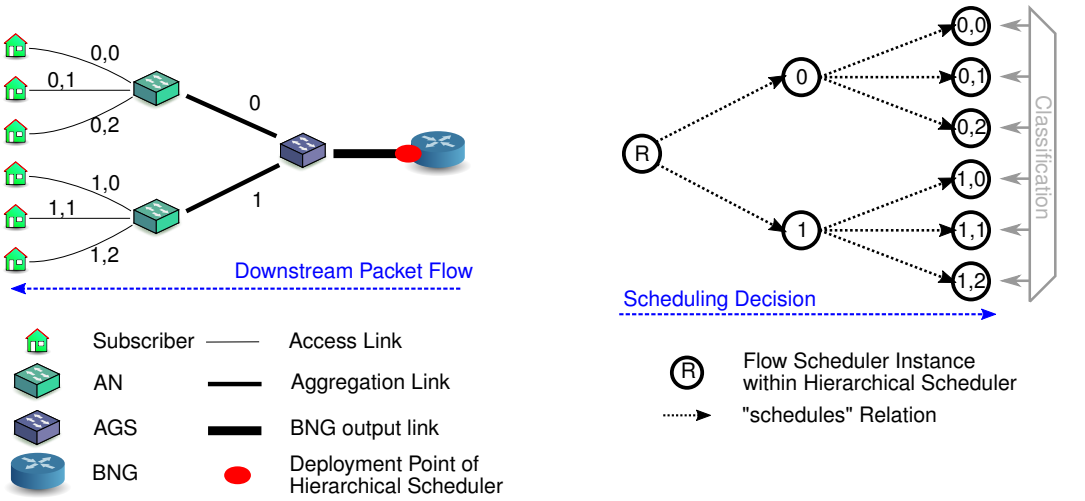
At a certain hierarchy level the ISPs typically insert a device called service edge router, edge router, Internet Protocol (IP) edge device or just IP edge that fulfills various crucial functions. A core function is downstream QoS enforcement, which requires this node to be placed before any potential bottlenecks.
In former architectures, these edge routers have been called Broadband Remote Access Server (BRAS) as in [TR-59] and often have been placed at the transition between aggregation and core network. Such placements typically result in several levels of aggregation between the BRAS

interface and an access link. Today, service edge routers are usually called BNG as in [TR-101] and are usually placed between aggregation and access networks. In some documents, the name-distinguishing feature is the basic network technology like Asynchronous Transfer Mode (ATM) for the BRAS in [TR-59] and Ethernet for the BNG in [TR-101]. Another commonly used term is Network Access Server (NAS) as used in [RFC5851]. Nevertheless, the core functions are the same and moreover, the terms are not used consistently in the community. In this document, we use the term BNG without implying that this device must serve ANs directly. We use the term regional access network for a network downstream of the BNG regardless of the number of hierarchy levels it consists of. A BNG typically possesses several downstream interfaces, each serving a regional access network that is isolated from the remaining ISP's network except for the connection by this interface. Links downstream of the BNG are often not redundant to simplify access control. In that case, placing the BNG topologically close to the access also means minimizing the impact of failures of physical links or interfaces. The BNG function may also be deployed deeper in the network and we explicitly consider a placement further up in the hierarchical topology in RADICCO.
BNGs usually fulfill several core functions for the subscribers served by it. A BNG implements Authentication, Authorization and Accounting (AAA) functionality, i.e. they authenticate the subscriber, provide authorization for service access and, if applicable, also cover accounting. A BNG also fulfills networking functions such as address assignment, routing and possibly Network Address Translation (NAT). It also works as Policy Enforcement Point (PEP), i.e. it enforces subscriber-specific policies, e.g. a maximum data rate even if served by a faster access link.

Another crucial function of the BNG is QoS enforcement. Today's networks not only transport traffic with different requirements and guarantees on aggregation links, they also provide a set of services to its private customers, typically the triple-play combination of VoIP, video streaming and Internet access. Usually, all three types of services are implemented on top of IP and transported in one packet-switched logical link on the access link. Nevertheless, since these services have certain QoS requirements to provide acceptable QoE to the user, the respective traffic has to be handled accordingly. From the ISP's perspective, the broadband Internet traffic is BE traffic, which means it is of lowest priority. Enforcing QoS guarantees in the access network requires proper classification and then appropriate queuing or scheduling [1]. To not require QoS guarantees at several places in the network, two contrasting approaches are used:

1. Overdimensioning in the core network.
2. Hierarchical scheduling at the service edge.

First, to not require QoS enforcement at fast core network nodes, the resources in the core network are lavishly dimensioned, so that any bandwidth shortage can only occur at the ingress or egress. Usually, ISP networks are designed to allow bandwidth shortages, i.e. bottlenecks, only in the access networks and such design is also confirmed by measurements [2, 3].
Second, to not require functionality for classification and scheduling in the lower network nodes as well as to not require these nodes, e.g. the ANs, to access AAA and policy repositories, the BNG implements hierarchical packet scheduling and traffic shaping on each downstream interface. This means that the scheduler of a BNG downstream interface implements all QoS-relevant functions in place of the downstream nodes so that these just require packet buffers. As detailed in Section 2.2, these scheduling and shaping functions may be integrated or performed by a hier-



(a) Exemplary topology (b) Corresponding hierarchical scheduler

Figure 2.2: Mapping of exemplary topology to hierarchical scheduler

archical scheduler and a separate hierarchical traffic shaper. In both cases the resulting function represents all downstream links and their capacities from the BNG downstream interface to the downstream access links in a reverse hierarchical structure as shown in Figure 2.2. Today, a typical router intended to be deployed as BNG can serve about 50.000 subscribers, but more important, today's products like the Nokia 7750 SR [4] or the Cisco ASR 9000 [5] typically use 1 or 10 Gb/s Ethernet downstream interfaces. In 2016, the average downstream speed in Germany is reported to be 14.1 Mb/s [6] and many ISPs offer broadband Internet access with 50, 100 or even 200 Mb/s. This combination means that on average, a BNG downstream interface can serve not more than 70 subscribers at their nominal access link speed.

The actual topology of the regional access network, i.e. how the subscribers served by one BNG downstream interface are connected to it, heavily depends on the technology of the access links. Therefore, we will give a short overview on today's technologies in that field and draw conclusions on access topologies of the near future.

2.1.2 Wired Access Technologies and their Impact on Topologies

Today, residential subscribers are connected to their ISP's infrastructure by either two-wire copper lines, a coaxial cable or an optical fiber. In Germany, by far most of the subscribers (2016: 75 % of a total of 31.2 M broadband subscribers [7]) use copper lines running one of the DSL variants. A significant and in the last years increasing number of subscribers (2016: 22.8 %) are connected by coaxial cable or short coax which is commonly operated applying the Data Over Cable Service Interface Specification (DOCSIS) system, at the time of writing typically DOCSIS 3.0. A rather small share of 2.2 % is connected using optical fiber, applying either a Point-to-Multi-Point (P2MP) technology such as Passive Optical Network (PON) or Point-to-Point[1] technologies such as Ethernet.

[1]Point-to-Point is often abbreviated by P2P but we use this abbreviation for Peer-to-Peer

For copper two-wire lines, the deployed generation of DSL impacts the maximum distance to the subscriber and therefore the number of subscriber lines terminated in a DSLAM. Basic Asymmetric Digital Subscriber Line (ADSL) ([G.992.1]) provides a maximum speed of 8 Mbit/s and a reach of, depending on the installed cable's attenuation and crosstalk, about 6 km. Newer generations, notably ADSL 2 (ADSL2) ([G.992.3]), Extended bandwidth ADSL2 (ADSL2+) ([G.992.5]), Very High Speed Digital Subscriber Line (VDSL) ([G.993.1]) and VDSL 2 (VDSL2) ([G.993.2]), in comparison increase the maximum bitrate for short lines (24 Mbit/s maximum for ADSL2+ and 250 Mbit/s for VDSL2). Nevertheless, in each case the increase compared to the predecessor technology only holds for ever shorter lines: ADSL2 and ADSL2+ increase bandwidth mostly for lines shorter than 2.5 km compared to ADSL, VDSL compared to ADSL2+ is faster only for lines shorter than 1.5 km. Because of this, the VDSL variants require the DSLAM being installed close to the subscribers. Usually it is installed in a cabinet at the curb, then also called outdoor DSLAM, and is connected to the AGS or BNG by fiber. Therefore, these concepts are also termed Fiber To The Cabinet/Curb (FTTC), highlighting that these technologies are a means to drive the optical network closer to the subscribers. The upcoming Fast Access to Subscriber Terminals, ITU G.9701 (G.fast) ([G.9701]) and its proposed successor XG-Fast [8] will improve bandwidth only for even shorter lines, 300 or 70 meters respectively. The decreasing distance also results in a decrease in the number of subscribers served by one AN, i.e. DSLAM, and an increasing number of ANs served by one BNG. Therefore, the resulting topologies for two-wire copper access networks often contain several layers in the aggregation network and rather few subscribers per AN. Today, there are also network architectures that use BNGs with a high fan-out and directly attached DSLAMs. Nevertheless, it is not clear if this concept is efficient if the ANs move even closer to the subscribers with G.fast or if hierarchical access networks will rise again.

Other access technologies, notably Hybrid Fiber Coax (HFC) that uses coaxial (TV) cables for the access link, but also Fiber To The Home (FTTH) / Fiber To The Building (FTTB) benefit from better physical channels. Therefore, in such networks the transmission speeds have been substantially increasing in the last years without reducing the practical reach by the same factor. For instance, a HFC optical node, the AN translating the optical signal to the coax cable, today can serve 25–2.000 subscribers. It typically serves several hundred subscribers (275 according to [RFC6057], > 300 according to [9]). So, the fanout degree of ANs of these networks is high compared to today's DSL topologies and can be expected to remain high. For HFC networks, DOCSIS 3.1 defines a transmission speed in the shared medium of 10 Gb/s [DOCSIS] and is being rolled out right now. A similar situation applies to PONs: 10-Gigabit-PON (XG-PON) was standardized in 2012 / 2016 [G.987, G.987.2], yet there is no significant deployment. Maybe it never will achieve large deployment since its successor, 40-Gigabit-Capable PON2 (NG-PON2) [G.989.2], might be used for future deployments.
All these developments show that for these technologies there is still room to serve many subscribers by one AN, so there is no need to introduce further hierarchies for physical reasons. In contrast to DSL, the technical platform of these technologies allows serving all subscribers with the same (maximum) capacity. Nevertheless, usually different capacities are sold and implemented to offer an appropriate service at an acceptable price to as many customers as possible (service differentiation). Moreover, all presented access technologies share that the upstream channel is configured with less capacity than the downstream channel.

Summarizing, today there are network architectures deployed that have just one hierarchy level downstream of a BNG interface, as well as architectures that have several levels of distribution beyond such a BNG interface. Moreover, it is very likely that both types of networks will continue to exist in the foreseeable future. Generally, access links are asymmetric in terms of bandwidth while the ISP's inner infrastructure uses symmetric technologies.

2.1.3 Traffic Patterns and Link Dimensioning in Wired Access Networks

Since most traffic in wired access networks is triggered by human actions, there is a correlation between the number of active users and traffic volume: The more people sleep or are not at home, the less traffic. Therefore, wired access networks show a clear diurnal load pattern [10, 11] with peak periods at the evening. The evening peak load even increased with the rise of VoD service operators such as Netflix, Amazon Video or Maxdome. A diurnal load pattern is also visible in the Internet in general [12, 13], but the extent is less strong in some regions [12]. The amount of daytime-invariant traffic may rise with the emerging Internet of Things (IoT) triggering transmissions without human interaction. Nevertheless, the volume of transmissions destined for a human receiver will also continue to increase, e.g. because VoD services switching to higher resolutions, 3D or higher color depth. Therefore, this imbalanced traffic pattern can be assumed to exist also in the near future.

Since each subscriber acts independently from its neighboring subscribers, the sum load of one AN and therefore its feeding link can be understood as a statistical function. So, the expected peak load of an AN is much lower than the sum of the subscribers' capacities. This allows the ISP to dimension his network accordingly and save costs. The concept of connecting and selling more overall capacity than available is called oversubscription. The ratio of the potential maximum demand, i.e. the sum capacity of all served links, to the actual bandwidth of the feeding link is called oversubscription ratio or contention ratio. Network architectures usually implement contention ratios higher than one at any aggregation level to save costs. At any node with a contention ratio greater than one congestion is possible. Nevertheless, oversubscription does not have to impair the service of any subscriber at all, if the underlying load distribution is predicted correctly. Unfortunately, this is a difficult or impossible task: Subscribers have different behavior and both subscribers as well as services change their behavior unpredictably over time. Moreover, ISPs tend to use optimistic forecasts or, in other words, are reluctant to upgrade their infrastructure when traffic grows. The reason is that companies need to provide a cost-efficient solution to their customers' needs at competitive prices.

So, the combination of heavily varying load and generous oversubscription results in access networks often not being able to transport all demands during peak periods. Since today's traffic is elastic on the packet level due to the use of CC, this usually does not mean services being unavailable or interrupted, but the users' QoEs being reduced. The meaning of reduced QoE depends on the type of service and varies e.g. from longer response times for web browsing to reduced video quality for VoD services. While this is hard to measure on the service level, the root cause, the reduced bandwidth available to a subscriber, can be measured easily. ISPs do not publish such information themselves, but there is a vast amount of indirect proof that available bandwidth is often significantly reduced during peak periods: In some rare cases there is measurement data available, for instance on the ISP Roger [2] or as the average of several ISPs.

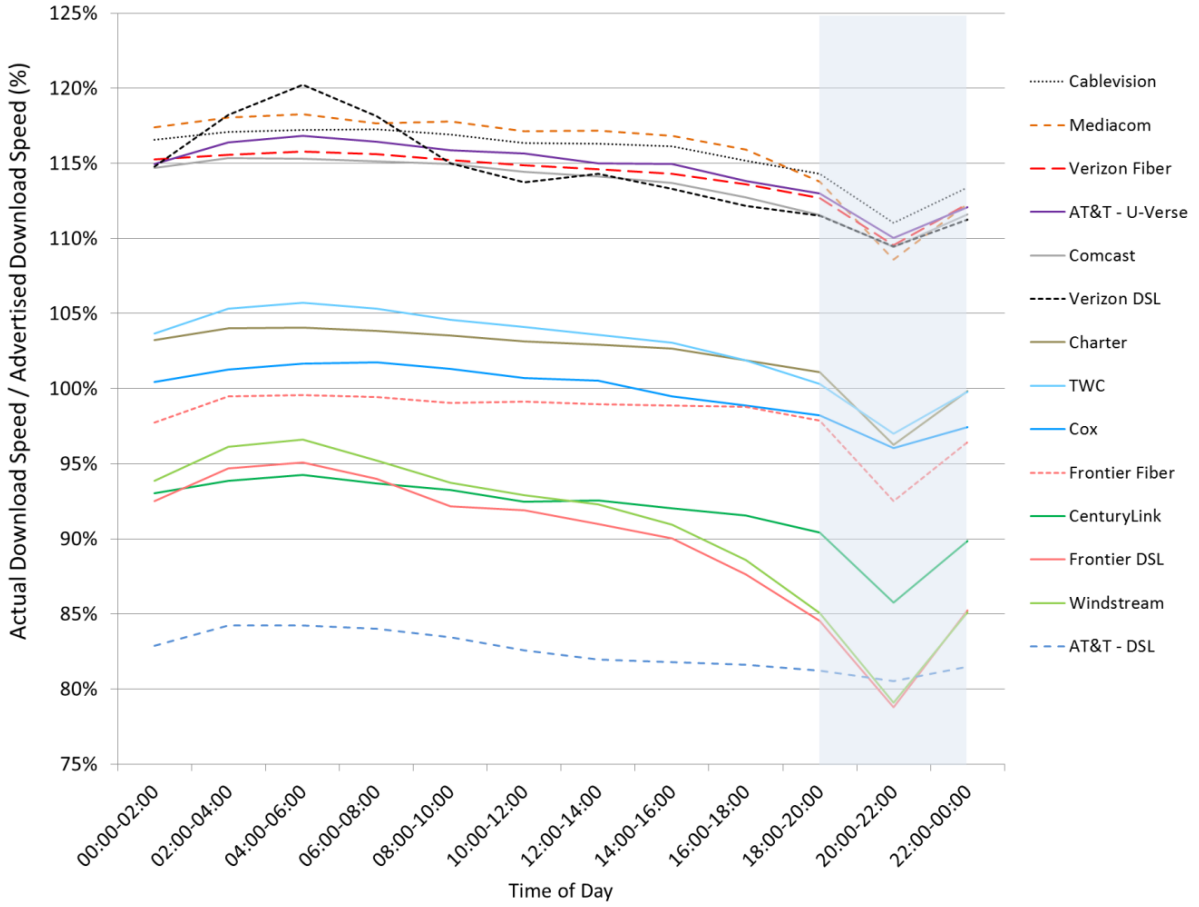


Figure 2.3: The ratio of actual download speed to advertised download speed during weekdays for two-hour time blocks (from [15])

In the United States of America (USA) as well as in Germany public agencies examined the broadband performance. The latest analysis of the German Bundesnetzagentur (BNetzA) [14] showed a significant decrease in downstream speed only for cable networks. In contrast, the more recent measurements of the Federal Communications Commission (FCC) for 2015 [15] show a clearer and alarming situation: As Figure 2.3 shows, the downstream transmission speeds of all examined ISPs are deteriorated during the peak period between 7:00 pm and 11:00 pm. A further indication that access networks cannot satisfy all demands during peak periods is the existence of the various peak load management approaches and products, see also Section 3.5 on related work on peak- and overload management. Such solutions basically delay certain transmissions and thus shift some load to less loaded times. Be aware that the networks of the measurement mentioned above usually already apply such technologies and still suffer from capacity shortages at peak periods.

Summarizing, traffic of private residential subscribers varies heavily with a diurnal pattern. For economic reasons, ISPs factor in multiplexing gains on medium and long time scales and therefore use oversubscription on every topology level in access networks. This frequently impairs the service provided to the subscribers during peak periods since there is a substantial amount of time during peak periods with higher demand for bandwidth at aggregation links than available.

2.2 Packet Scheduling

This section gives an overview on the state of the art in packet scheduling with focus on wired access networks. It starts with an introduction and definitions (Section 2.2.1), followed by an summary of the evolution in packet scheduling research (Section 2.2.2). In continuation, performance of packet schedulers is presented, in detail regarding fairness (Section 2.2.3) and regarding complexity (Section 2.2.4). Further, schedulers meeting special requirements are discussed: hierarchical schedulers (Section 2.2.5), rate limiting schedulers (Section 2.2.6) and schedulers for multi-class traffic (Section 2.2.7). Lastly, this background is mapped to the focus of this thesis, the BE packet scheduling at the edge of access networks (Section 2.2.8).

2.2.1 Introduction and Definitions

Packet scheduling is part of the link layer in the OSI Model [X.200], more precisely of the MAC sublayer. Packet scheduling denotes the process or the function at a packet switched network interface that

- Decides which packets are sent and when, and which are buffered.
- Aims for some performance guarantees referring to well-defined subsets of the incoming packets.

The well-defined subsets of packets are commonly called flows in scheduling context. Usually, any incoming packet can be mapped to exactly one of the flows. This definition does not match the definition of flows in the context of layer 4 protocols such as TCP. The objective of packet scheduling algorithms is usually enforcing QoS guarantees on the flows. The exact objective varies heavily from use case to use case. For wired networks, overall QoS improvement is achieved by prioritizing traffic classes with QoS guarantees, e.g. carrying VoIP traffic, at BE's cost, and by improving or enforcing fairness amongst the flows within a class [16]. For wireless networks, there are more parameters that can be considered, most prominent the current capacities of the channels to the different receivers (due to changing physical attributes). In this work, we focus on packet scheduling for wired access networks, i.e. for schedulers for channels with static properties.

It is crucial to distinguish packet schedulers from the closely related queuing disciplines. Queuing disciplines also are MAC functions that decide which packets are sent and when, and which are buffered. Yet, queuing disciplines aim for objectives regarding the aggregate. So, a queuing discipline is defined by its enqueue decision function and its dequeue decision function that manage a packet buffer.
Consequently, queuing disciplines work on packets without distinction, while packet schedulers operate on flows.

Queuing disciplines that implement a non-trivial algorithm for buffer management are also called Active Queue Managements (AQMs). There is a huge variety of AQMs, many of them targeted for specific scenarios. The decision logic of most AQMs and all that are in today's high-speed routers and switches, is implemented in the enqueue decision function only. These AQMs use a trivial dequeue decision function equivalent to retrieving the first element of the

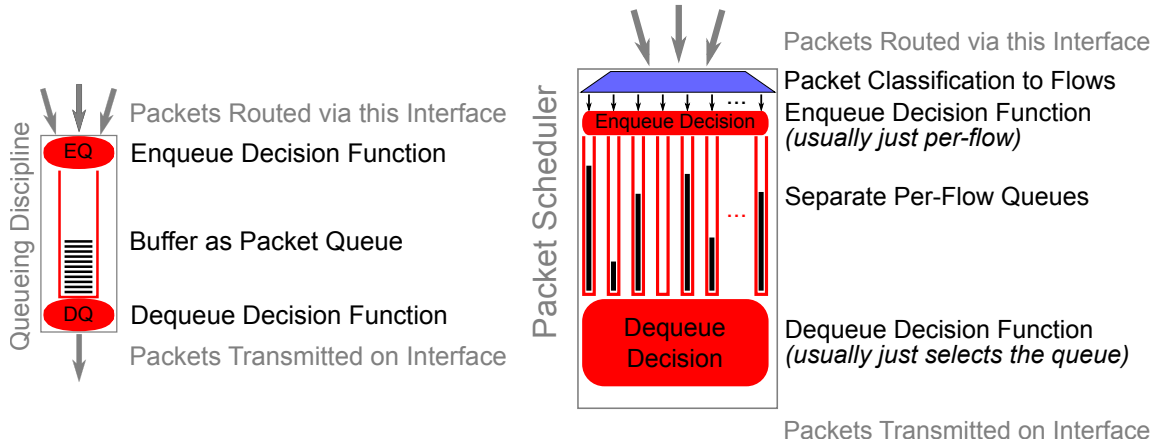
queue data structure (often a linked list or a ring buffer in case of an array-backed buffer). In today's packet switched networks it typically makes a difference, which packet is served when on a link. This is of special importance if the traffic is composed of self-contained subsets, e.g. of packets of different subscribers, different services or different connections. Scheduling not only directly impacts packet delays by internally using packet buffers, it also decides on the flows whose packets are rejected due to full buffers because it uses a separate queue for each separately scheduled flow. So, scheduling is an important place to implement QoS. This is a important reason among others, why queuing disciplines are not sufficient at many places in a network [RFC970, 17].
These definitions, starting with the clear distinction between packet schedulers and queuing disciplines, are not consistently used in neither research nor industry. For instance, packet schedulers and queuing disciplines are called classful queuing disciplines and classless queuing disciplines in the Linux kernel.

Technically, both packet schedulers as well as queuing disciplines can globally be described as an Enqueue-Decision-Function deciding which packets are accepted for transmission at all, and a Dequeue-Decision-Function deciding which of the buffered packets is transmitted next. Queuing disciplines use just one global instance of a decision algorithm and one universal packet buffer or queue as depicted in Figure 2.4a. In contrast, packet schedulers use per-flow queues and work based on three components as depicted in abstract illustration in Figure 2.4b:

1. A classification function maps every incoming packet to its corresponding flow.
2. An enqueue decision function decides based on the classification context if the packet is accepted and buffered.
3. A dequeue decision function decides when and which flow is dequeued, i.e. to which flow bandwidth is assigned at that moment.

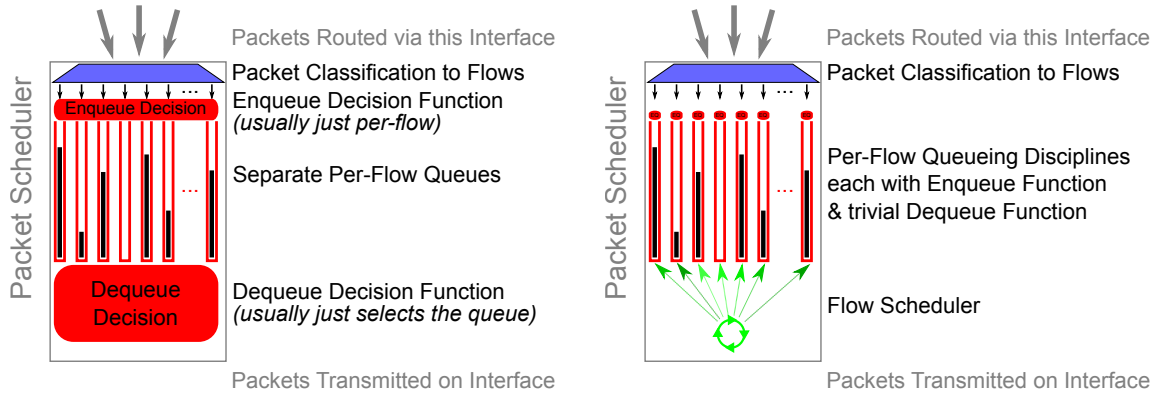
Occasionally, this last function is called packet scheduler. This scheduler schedules packets for transmission on the respective link, but this selection is not based on the packets themselves but taken on flow-level granularity and depending on the flows' states. Therefore, we use the term flow scheduler for this function or, when there is no risk of confusion, just scheduler.

For many packet schedulers, these three components—classification, flow-queue management and selection of a flow for dequeue—are independent. This is beneficial since for software implementations as this allows a modular design or even dynamic configuration of packet scheduling and therefore applying packet schedulers tailored to the use case and the operator's goals. For example, the packet scheduling in a regional AN should not depend on whether the operator assigns only an Internet Protocol version 4 (IPv4) address to a customer, or an IPv4 address and an Internet Protocol version 6 (IPv6) address (so-called dual-stack operation) or just an IPv6 address using Dual-Stack Lite (DS-Lite) [RFC6333]. In case of modular packet schedulers, the management of each per-flow-queue is equivalent to a queuing discipline, that means here the flow scheduler can be combined with any queuing discipline or AQM. Similarly, the dequeue decision function usually does not take any state of the queuing disciplines into account, but is just a flow scheduler that selects the flow to be dequeued among the non-empty queuing disciplines and calls its dequeue function. It must be noted that the Dequeue-Decision-Function of queuing disciplines usually just consists of the `POP()` operation of a First In, First Out (FIFO) list, a very cheap operation in terms of complexity and absolute processing cycles. Nevertheless, there exist AQM algorithms that define complex and potentially costly operations



(a) Structure of a queuing discipline (b) Generic structure of a packet scheduler

Figure 2.4: Comparison of the structures of queuing discipline and packet scheduler



(a) Generic structure

(b) Typical structure of classifier, queuing disciplines and flow scheduler

Figure 2.5: Comparison of a generic packet scheduler structure and the typical structure

on the packet dequeue. The most prominent example in recent times is Controlling Queue Delay (CoDel) [18]. To date, these AQMs are not used in packet schedulers of network operator equipment because of the computational effort necessary at the time-critical dequeue event.

Figure 2.5 shows the difference between the generic structure (Figure 2.5a, same as Figure 2.4b) and the typical structure of deployed packet schedulers consisting of classifier, queuing disciplines and a flow scheduler (Figure 2.5b).

Packet Classification

Usually, classification is carried out based on simple labels on low layers, e.g. (destination) IP addresses and Differentiated Services Code Point (DSCP) values, and combinations thereof. In many cases, the incoming traffic is already appropriately labeled, e.g. since the ISP's video streaming servers mark their packets accordingly. Packet classification may also base on arbitrary

complex functions because the classification function can be separated and distributed to several external nodes. If a complex classification is carried out separately, this function must insert other, simple labels to transfer the information to the actual packet scheduler. DPI-based overload management, see Section 3.5.4, may be viewed as implementation of such a separated classification function: The DPI function aims to identify and mark non-urgent traffic in a separate node and the BNG's flow scheduler serves these flows with lowest priority only. This example shows that external classification functions may be complex, require special hardware and may also be managed and even replaced independently, thus rendering the whole system more flexible.

In hierarchical schedulers, packet classification represents that hierarchy, e.g. for ANs there are not only attributes mapping to subscribers, but also attributes mapping to privileged video streaming or the VoIP class. Nevertheless, the combination of attributes finally identifies any packet to be queued into one specific queue, so packet classification may also be perceived as a flat mapping even for hierarchical schedulers. For BNG packet schedulers at BNG downstream interfaces two levels of classification can be distinguished: identification of the subscriber and identification of service class. Mostly, subscriber classification in BNGs is performed based on the destination IP address since usually network operators assign one or two addresses (IPv4 and/or IPv6) to each subscriber. Some operators do not own enough addresses so they use NAT or Network Address and Port Translation (NAPT). In that case, packet classification for subscriber identification is performed using destination IP address and port range. The identification of prioritized traffic, typically the ISP's VoIP and video services, could be performed on source IP address ranges and ports, but probably is based on labels already attached upstream of the BNG, maybe even at the source. Candidate labels are DSCPs, Virtual LAN (VLAN) tags or Generalized Multi-Protocol Label Switching (GMPLS) tags, depending on the deployed technology.

The concept and the algorithm presented and examined in this thesis does not touch this part of packet scheduling. It also does not depend on packet classification specifics and thus can be combined with any packet classification mechanism desired.

Queuing Disciplines

For every flow served by a packet scheduler, the scheduler maintains a queue of packets[2]. Usually, packet schedulers use just one type of queuing discipline for all queues at a time although this is not technically necessary. Furthermore, typically there are only simple and well-tested queuing disciplines configured, namely either a simple bounded FIFO queue, also known as tail drop queue, or a Random Early Detection (RED) queue [19]. The queuing discipline is the major enforcement point for delay-related guarantees since the queuing delay can be much bigger than the delay induced by the flow scheduler. Nevertheless, for BE traffic delay bounds are either lax to allow full bandwidth utilization (see Section 2.4.4) or not specified at all.

As in stand-alone deployment, the goal, and therefore basis for assessment, of any queuing discipline in packet schedulers is twofold:

[2]Some implementations discard empty queues and recreate queues once a packet arrives.

- The AQM should avoid a standing queue, i.e. avoid the minimum queue size being (significantly) larger than zero. A standing queue has no advantages, but induces undesired delays on application layer as well as in the transport layer control loops.
- The AQM should avoid wasting bandwidth by unnecessary packet drops. Specifically, it should avoid dropping packets in a way that may trigger senders to reduce their sending rate so that the flow's queue is empty for a significant time share and thus less bandwidth than possible is allocated to the flow.

In case of a packet scheduler, achieving these opposing goals of low delay and high utilization also depends on the flow scheduler and of course on the incoming traffic. Nevertheless, many CCAs require significant buffer to achieve high utilization (see Section 2.4.4), so the AQM and the effective buffer size is a configuration parameter important for the packet scheduler's performance.

Buffer sizing as well as the choice of queuing discipline for a packet scheduler are questions to be answered independently from honoring end hosts' CC-based prioritization in BNG packet scheduling. The concept and the algorithm presented and examined in this thesis do not depend on queuing discipline specifics and thus can be combined with any queuing discipline. Nevertheless, the presented approach requires buffer sizes to be large enough to detect if a CCA makes aggressive use of available buffer to distinguish foreground from background traffic. As argued in Section 2.4.4, buffer sizes on subscriber level cannot be expected to substantially decrease in the near future.

Flow Scheduler

Having said that neither packet classification nor a packet scheduler's queuing disciplines are in focus of this work, obviously, the flow scheduler is. This component is also in focus of most research and publications on packet scheduling. Many publications carrying packet scheduling in the title cover the flow scheduler only, often the term *packet scheduler* is even used for only the *flow scheduler* component.

In fact, many publications ignore the other components and by that culpably neglect their impact on overall performance. Moreover, many evaluations base on theoretical traffic models or even assume a static set of backlogged flows. This disregards that today's Internet traffic on the packet level does not follow stochastic models, but is mostly controlled by transport layer CCAs. Yet especially for such traffic, the queuing discipline has significant influence on the packet scheduler's performance regarding both delay as well as throughput (see also Section 2.4.4). Moreover, both packet classification and queuing disciplines contribute to the overall packet schedulers computational and hardware requirements. If chosen poorly, their share may well exceed the cost of the flow scheduler. A proper choice of mechanism and implementation however allows to limit the computational effort for any incoming or outgoing packet in these components to a fixed number of cycles which is independent of the size of the overall task, i.e. the number of flows managed by this packet scheduler. Computer scientists denote the complexity of these algorithms being *O(1)* in Bachmann-Landau notation, the lowest possible and most desirable complexity class. For the flow scheduler, much research on flow schedulers

targets exactly this topic, reducing the asymptotic computational complexity, often trading off accuracy in the guarantees enforced. Here a note is in order: asymptotic complexity is an important metric, the higher the number of flows, the more significant. In contrast, if the number of flows is known for some use case in advance (or at least its order of magnitude), the computational effort of algorithms can be compared for that specific use case. This is most relevant for this work. For example, Deutsche Telekom are migrating their network structure to a network architecture named *TeraStream*. This architecture means that one BNG interface serves not more than 250 subscribers and distinguishes four traffic classes per subscriber [20]. This results in maximum 1.000 separate flows in the packet scheduler, so *O(log N)* is probably a well acceptable complexity since it results in only ten basic operations (typically comparisons) per scheduling event. Assuming a per-packet scheduling, an interface speed of 10 Gbit/s and an average packet size of 1000 Bytes, this results in 1.25 M dequeue events per second and 12.5 M operations per second. This is clearly feasible to implement.

The flow scheduler is the enforcement point for bandwidth-related policies and performance guarantees. It also contributes to delay-related guarantees since the flow scheduling results in a statistical delay in addition to the queuing delay. But packet delay is not of interest in case of BE traffic, the class this thesis focuses on. The performance guarantees aimed for by the packet scheduler may relate to all flows or just a subset, and may be relative or absolute. Practical examples are:

- All active subscriber flows of class X shall receive the same bandwidth at any point in time.
- The VoIP flows shall receive minimum delay.
- The flows of class Y, e.g. the business customer flows, shall receive a minimum bandwidth of Z Mbit/s.

These examples require an appropriate classification of incomings packets along with meta information to define the groups of flows, e.g. which flows correspond to business customers. Second, to express absolute guarantees the available resources, i.e. the available bandwidth in that case, obviously must be reliably sufficient to satisfy all guarantees at the same time, else only statistical guarantees can be given. Third, often scheduling guarantees only make sense in combination or combined with knowledge of properties of the incoming traffic, e.g. that any VoIP flow is not more than 200 kBit/s. These properties are often enforced prior to entering the packet scheduler, e.g. by applying rate shaping based on leaky bucket or token bucket approaches.

2.2.2 Packet Scheduling Evolution

In this section, we will give an overview on the development of scheduling algorithms, give information on how and why schedulers are used in networks and finally shortly present the two main scheduler families relevant today: schedulers based on Deficit Round Robin (DRR) and based on or similar to Weighted Fair Queueing (WFQ) that use timestamps. A good overview on algorithms can be found in Dordal's book [21], focused on the algorithms' properties and use cases rather than complexity.

The need to make resource allocation decisions on a packet-basis came up with the emergence of Virtual Circuits (VCs). A VC requires explicit setup like a circuit, but VC networks are packet switched networks using statistical multiplexing and therefore can benefit from statistical multiplexing gain. This means that routing decisions have to be taken not for every packet but only once per VC. But it also meant a fundamental change in resource allocation. Traditional circuit-switched networks use static resource allocations which were explicitly reserved during circuit setup, e.g. time slots on any interface on the path of a circuit for Time-Division Multiplexing (TDM) systems. In contrast, for VCs there arises the need to take resource allocation decisions on a per-packet basis whenever multiplexing several VCs to one network interface. So, algorithms have been designed that served the purpose satisfactorily. For example, the *TYMNET* architecture [22] used one FIFO queue for each VC and served the queues in a round robin manner (while not using the term round robin). Already this algorithm supported weights insofar as priority channels receive more than one turn per round. The *DATAKIT* system proposed by G. Riddle, which is also based on VCs, uses per-VC FIFO queues too, but manages these queues in two lists [23]. One list contains VCs having had no packet in its queue at arrival of the VC's last packet. So, it contains VCs with newly starting transmissions and VCs with a load lower than the VC's current possible share on that link. The other list contains VCs having had at least one packet in its queue at that time, i.e. the VCs with ongoing transmissions of higher load. While there are entries in the first list, these queues are served in a First-Come, First-Served (FCFS) manner (alias to FIFO), once it is empty, the queues in the second list are served, also in FCFS sequence.

While for VC-based networks the packet classification is predetermined, this is not the case for packet switched networks. Usually also for packet-switched networks a packet scheduler is desired instead of one trivial FIFO queue: The FIFO strategy is perceived being inherently unfair since it allocates most resources to the fastest sender. Nagle therefore introduced the concept of fairness [24], interpreted as fairness between source nodes: any source shall receive the same bandwidth at an interface as any other source currently using this interface. This goal is achieved by a packet scheduler maintaining one FIFO queue for every source node in the network and a flow scheduler serving the non-empty queues in a round robin manner. So Nagle transferred the existing concept of a packet scheduler from VC networks to packet switched networks. It should be noted, that the goal of this packet scheduler is a purely relative one: there are no guarantees on maximum delay or minimum throughput for any flow, i.e. any traffic source.

At about the same time network researchers started to apply existing theoretical models, insights and algorithms from other scheduling research, in particular operations research, to packet scheduling. The most important contribution from operations research is the Generalized Processor Sharing (GPS) model, proposed by Kleinrock [25, 26] and analyzed by himself and others [27, 28]. This research reduces the packet scheduler to bandwidth assignments, leaving out potential other goals such as limited delay. So this research community reduces a packet scheduler to a queue selection algorithm aiming for distributing bandwidth among flows as close to predefined shares as possible. This task directly corresponds to the task of assigning jobs to be processed by a processor, each job having a priority. So processing jobs correspond to packets, and priorities correspond to the processing share to be received by jobs of this priority.

Assuming a job (or packet transmission) may be interrupted at any time without cost, the GPS model defines a time-sharing system with the slot time approaching zero. So jobs are "cycling

around at an infinite rate, receiving an infinitesimal quantum of service infinitely often" [25]. In this theoretical system, at every point in time every job received exactly its correct share, i.e. GPS achieves perfect proportional fairness. Note that this fairness criterion differs from other fairness definitions, such as max-min fairness [29], which is aimed for by some schedulers for wireless networks (on objectives in bandwidth sharing see also [30]/[31]). Since packets can only be transmitted as a whole, in real packet schedulers the predefined shares cannot be exactly met at most points in time. Nevertheless, GPS defines the perfect reference for packet schedulers with regard to bandwidth distribution. Moreover, queuing theory could be used to derive delay distributions to be expected for several packet schedulers. E.g. Fraser derived delay distributions [32] for the packet scheduler already presented in [23] and Lo derived delay formulas for the packet scheduler implemented in AT&T's Trunk Interface Module (TIM) [33]. Both, the ideal scheduling of GPS as well as proven traffic properties, are since core research topics in packet scheduling.

When assessing scheduling algorithms, i.e. algorithms for the flow scheduler within a packet scheduler, usually two domains are of interest: the computational complexity and the quality of worst case schedules. This quality is mainly defined by two aspects: First, by the schedule's long-term fairness, thus its deviation from the ideal schedule of GPS. Second, by the schedule's burstiness, i.e. its short-term fairness.

More formally, schedulers are examined in three crucial metrics:

Complexity The worst-case number of computational steps needed for selecting the next flow. It is usually given in Bachmann-Landau notation, typically dependent on the number of flows.

GPS-relative delay The worst-case delay of a packet compared to the ideal completion time with GPS. It also is usually given in Bachmann-Landau notation. Note that this criterion implies aiming for proportional fairness.

Fairness Several metrics are used to capture different aspects of fairness. The most important fairness metric is its normalized Worst-case Fair Index (nWFI).

These metrics will be introduced in detail in the next sections.

2.2.3 Delay Relative to the Perfect Schedule and Fairness of Scheduling Algorithms

Before starting with the GPS-relative deviation or delay, we introduce some terms and make some general remarks.

When describing schedulers and their produced schedules, the transmission times of packets are of interest, more precisely: The Finish Times of these transmissions and sometimes also the Start Times. The *Start Time* S_i^j denominates the point in time the first (infinitesimal) piece of the i-th packet of the j-th flow is transmitted on the scheduled interface. Correspondingly, the *Finish Time* F_i^j of the i-th packet of the j-th flow denominates the point in time its transmission on the interface is completed and the interface is ready for the next transmission. In the networking

context, a comprehensive description of a schedule is defined by the Finish Times of all packets. Note that most theoretical analyses assume a constant set of backlogged flows, i.e. there neither are flow queues running empty nor are queues becoming non-empty, i.e. backlogged. Moreover, most schedulers are designed to be *work-conserving*, that means the managed link is only left idle if there is no packet of any flow available. All schedulers discussed in this section belong to that category. Therefore, Start and Finish Times may be expressed in time units as well as in bits or bytes (for variable rate links the mapping function may be not constant but for sure is bijective). Note also that, although preemption, i.e. canceling an ongoing transmission in favor to start transmitting a newly arrived packet, may improve some metrics of a scheduler, this is usually not considered since it wastes bandwidth.

When comparing a packet scheduler with GPS, it obviously cannot achieve the same schedule in most scenarios because GPS uses infinitesimal short time shares while a real scheduler can only schedule full packets. The Bit-by-Bit Round Robin (BBRR) scheduler is another idealized reference scheduler, one step closer to reality but still theoretical: A scheduler that serves all active flows in a round robin manner, transmitting one bit for each flow in each round. Compared to GPS, this algorithm neglects that packets not only arrive when a transmission (of one bit) ends. Thus, the error in Finish Times is less than the transmission time of 1 bit. This error can be neglected, so often BBRR is used as reference instead of GPS.

A real-world packet scheduler can obviously neither achieve the same schedule as BBRR nor as GPS since the units of transmission, i.e. of service of the scheduler, have to be packets. But there are still significant differences in accuracy between schedulers, i.e. of difference between the schedule of that scheduler and the one of GPS. For example, Nagle's algorithm [24] described above, would result in a source sending packets twice as big as another receiving about twice as much bandwidth. So, here intuitively the unfairness is unlimited and so is the maximum difference between Finish Times of this scheduler and GPS. Moreover, the maximum difference of service between two flows of same weight is also unlimited. The main cause for this deviation is the calculation being based on packets, not on bandwidth or transmitted bits. Other schedulers do better, some of them explicitly aligning themselves to GPS. This is easily possible since the Start Times and Finish Times of GPS can be calculated and may serve as guidance for a scheduling algorithm.

Using time stamps can also be used for classifying schedulers: There is the group of timestamp-based schedulers and there is a group applying other approaches. Among the second group, Round Robin (RR)-based schedulers are the most relevant ones. First, we present timestamp-based schedulers.

Demers, Keshav, and Shenker were the first to publish such an algorithm [17] which they simply called *Fair Queueing (FQ)*, shortly followed by Zhang [34] who called her algorithm *VirtualClock* (more thoroughly examined in her PhD thesis [35]). Both are based on calculating the Finish Time a reference system would achieve for any incoming packet and then transmitting packets in increasing order of these so called *Virtual Finish Times*.

The reference for the timestamps of FQ is GPS, although the authors mostly argue using uniform weights and only shortly mention, how to extend the scheduling to arbitrary weights. Not focusing on different weights is owed to the motivation for this scheduler: Demers aimed for a bandwidth allocation to achieve protective and stable networks. In FQ Demers et al. added an

optional corrective factor to bring forward short packets but if set to zero as in their measurements, the resulting schedule is very close to GPS. Although the term Weighted Fair Queueing (WFQ) is not used in the paper itself, this term is widely used for the weighted algorithm proposed by Demers et al. This algorithm is also equivalent to a scheduling algorithm proposed by Parekh called Packetized Generalized Processor Sharing (PGPS) [36].

The focus of VirtualClock on the other hand is inspired by a reservation-based network such as a TDM network and aligns its timestamps to the virtual transmission times in such a network. Therefore, it is different in that it expects clock speeds, i.e. corresponding bandwidth assignments, to be explicitly configured for all flows. Because of that, it inherently supports different weights. Since it in contrast to its original inspiration is work-conserving, it could be shown that it in fact approximates GPS. Also Zhang considered an optional prioritization of short flows, but her approach only affects the beginning of a flow, so flows not fully utilizing their share would not persistently benefit as in case of the original FQ.

For WFQ, PGPS and VirtualClock the maximum discrepancy to BBRR for each flow was proven to be within P_{max}, denoting the maximum packet size [17], i.e. for each flow the number of bits already transmitted with one of these algorithms never differs more than P_{max} from the number of bits BBRR would have transmitted. This is also the minimum per-flow error for algorithms scheduling complete packets (without preemption).

Although this is the minimum error and despite the closeness to GPS suggested by the name PGPS, the schedule of WFQ is more unfair than necessary. Essentially, the WFQ schedule often exploits this maximum error where it could be avoided, especially for configurations with heavily differing weights, e.g. by one or more orders of magnitude. The resulting WFQ schedules also show oscillation patterns in bandwidth allocated to some flows, which is generally not desirable since oscillations cause problems e.g. for end-to-end control loops. As cause for this problematic behavior Bennett and Zhang identified flows not being serviced for quite a long time in WFQ while packets of other flows are serviced before they would even start receiving service with GPS [37, 38]. Note, that in GPS a packet arriving to an empty queue is serviced immediately, else without delay after finishing the packet before. By deciding on the schedule based on finish times, packets may receive service from WFQ although GPS would not have finished serving the packet before.

To capture this type of unfairness, Bennet and Zhang introduced worst case fairness and then the nWFI as a metric. A scheduler is worst-case fair, if for any flow j there exists a time constant D_j for which holds: A packet p_i^j arriving at τ will be served latest after the time the existing queue takes to be served with j's guaranteed rate r_j plus this delay constant D_j, i.e. Equation 2.1 holds. D_j is called the Worst-case Fair Index (WFI) of flow j.

$$S_i^j \leq \tau + \frac{Q_j}{r_j} + D_j \tag{2.1}$$

For meaningful comparison between flows, the maximum delays need to be normalized. A scheduler's nWFI is then defined as the maximum normalized delay D_j as shown in Equation 2.2. For further details, refer to [37, 38].

$$nWFI = \max_j \left(\frac{r_j * D_j}{\sum_i r_i} \right) \tag{2.2}$$

Summarizing, nWFI is defined as a normalized upper bound for the time between a packet's start time according to GPS and its start time for the scheduler being assessed. The nWFI by that describes the short-term fairness of a scheduling algorithm. So, a low nWFI guarantees resulting schedules being not very bursty (for a static set of active flows). For WFQ, Bennett and Zhang find that the nWFI scales with $N/2$, N being the number of flows.

Bennett and Zhang propose a new algorithm, named Worst-case Fair Weighted Fair Queueing (WF^2Q), that has the same delay bound as WFQ and a flow's WFI corresponds to the transmission time of a maximum sized packet at that flow's guaranteed rate. This means that the WFI does not depend on the number of flows but only on the share of that flow. This results in a nWFI that just corresponds to the maximum size packet transmission time, which is the lower limit since any non-preemptive packet scheduler must deviate from GPS by one maximum size packet in the worst case. They achieve this smoother scheduling by limiting the flows that may receive service to that flows whose first packet would have started to receive service in GPS at that time, too. They call the head-of-queue packets, that in GPS would already have started to receive service — and maybe even finished — *eligible packets*. Therefore, this selection strategy is called Smallest Eligible virtual Finish time First (SEFF).

So WF^2Q selects the next flow to dequeue based on both virtual start time and virtual finish time, so data structures sorted by both indexes are necessary. Although this is not welcome, all schedulers using only one characteristic have large WFIs as Stephens, Bennett and Zhang pointed out [39]. They developed a follow-up scheduling algorithm called WF^2Q+ [38] that uses two characteristics but reduces algorithmic complexity to $O(logN)$. WF^2Q+ results in the same schedule as WF^2Q and by that inherits WF^2Q's tight delay bound and its optimal nWFI.

The "Leap Forward Virtual Clock (LFVC)" algorithm [40] further reduces complexity. By coarsening the timestamps and using complicated data structures for the flows' timestamps it achieves an overall complexity of *O(log log N)*. But in contrast to WF^2Q+, LFVC cannot maintain the *O(1)* GPS-relative delay, it achieves only *O(N)* GPS-relative delay. Nevertheless, LFVC achieves an nWFI that is independent of the number of flows and only marginally bigger than the minimum possible value achieved by WF^2Q+.

All algorithms analyzed to this point explicitly select the flow to dequeue based on timestamps stored for each flow, e.g. virtual start and finish times. There is a second important group of scheduling algorithms that work fundamentally differently: the Round Robin (RR) schedulers. As the scheduler in TYMNET [22] mentioned before, algorithms of that class maintain a list of all active flows, i.e. non-empty queues, and traverse through this list. Basic RR schedulers such as the TYMNET scheduler that just dequeue a maybe weight-dependent but fixed number of packets from each active flow in a round robin manner are obviously not fair since flows with bigger packets receive an undeserved larger share that may grow without limits. The most wide spread method to compensate for different packet sizes is to manage state for each flow representing a right to send data and increasing this value with every round by a weight-depending quantum. Shreedhar and Varghese were the first to propose a scheduler based on this mechanism and called their algorithm Deficit Round Robin (DRR) [41]. DRR is based on a positive quantum of at least P_{max} (necessary for the complexity guarantee) and one state variable stored for each flow, the flow's *deficit*. This deficit represents the right to send data which could not be consumed yet. In each RR iteration DRR adds the fixed quantum to the flow's deficit variable. Then, the deficit counter is sufficient to send at least one packet by definition of the quantum. So, the deficit

counter is reduced accordingly and the first packet in the queue is sent. This is repeated until the available deficit is not sufficient or the queue runs empty. If the queue runs empty, the deficit counter is reset to zero and the queue is removed from the list of active flows. It was proven that the maximum deviation of this algorithm is 3*P_{max}. DRR has a nWFI of $O(N)$.

Summarizing, there is a limit for accuracy of packet schedulers, which is the delay bound of one maximum sized packet. There exist several algorithms that reach this maximum accuracy.

This limit is achieved by several scheduling algorithms but also within this group, schedules may differ significantly with respect to networking as a whole, defining a second order unfairness. The WFI is an accepted means to capture this unfairness, and WF^2Q and WF^2Q+ reduce this unfairness to the theoretical minimum, so both provide an optimal schedule by both metrics. Nevertheless, they require virtual start and finish times being computed and evaluated for all flows, so complexity is not trivial and deserves attention.

2.2.4 Complexity of Scheduling Algorithms

When analyzing complexity of basic scheduling algorithms, again the two large groups can be identified: On the one hand RR algorithms (e.g. [41, 42, 43, 44, 45, 46, 47]) that require very low and, most important, constant computational effort, i.e. complexity of *O(1)*, for selecting the next flow to be dequeued. These schedulers are fast but have non-perfect fairness and delay properties as outlined before. On the other hand, there are algorithms that manage timestamps for each flow and decide on the next flow based on this information (e.g. [17, 34, 36], [38] & [37], [48], [49], [40]). Examples for this group are WF^2Q and WF^2Q+, which as most of the algorithms of the second group aim to emulate GPS. For each dequeue event, any algorithm of this group needs to perform two operations: First, it needs to calculate a new timestamp for the served flow. The cost for this operation varies between *O(N)* for WFQ, *O(log N)* for WF^2Q+ and even *O(log log N)* for LFVC that uses sophisticated special data structures. Second, it needs to identify the next flow to dequeue. Most efficiently this is done by maintaining a priority queue of active flows and re-inserting the recently removed head of that queue by the updated timestamp(s). Such ordered insertion has a complexity of *O(log N)* on standard hardware, but may be implemented in *O(log log N)* under certain conditions as by LFVC [40]. Other algorithms achieve a complexity independent of the number of flows *N* by using rounded time stamps, e.g. [50, 51] but they pay a price by a non-optimal GPS-relative delay. Xu and Lipton proved that GPS-relative delay of *O(1)* requires a minimum complexity of *O(log N)* [52].

In contrast to GPS-relative delay, a good nWFI, i.e. < *O(N)*, can be achieved with lower complexity.

So generally, there is a trade-off between low latency, low complexity and fairness, which also has been looked into in the research community [53]. In consequence, some algorithms have been developed that use a hybrid approach, i.e. time stamps on one level and a RR mechanism on others, e.g. Fair Round Robin (FRR) [45]. Yuan and Zhenhai explicitly aimed for a low-complexity but fair scheduler when designing FRR. They achieved their goal by combining both classical scheduling concepts in a two-level scheduler: FRR uses a time stamp-based scheduler for scheduling between a small number of flow classes and a DRR-based one for scheduling the flows within these classes.

Table 2.1: Summary of scheduler properties.
N = number of flows, n = number of groups / levels / classes depending on algorithm

Scheduler	Concept	GPS-relative Delay	nWFI	Complexity
GPS [25]	theoretical	0	0	n.a.
WFQ [17]	timestamps	O(N)	O(N)	O(N)
SCFQ [49]	timestamps	O(N)	O(N)	O(log N)
VirtualClock [34]	timestamps	O(1)	∞	O(log N)
DRR [41]	round robin	O(N)	O(N)	O(1)
WF^2Q [37]	timestamps	O(1)	O(1)	O(logN)
WF^2Q+ [48]	timestamps	O(1)	O(1)	O(logN)
LFVC [40]	timestamps	O(N)	O(1)	O(log log N)
Stratified RR [44]	timestamps + RR	O(N)	O(N)	O(n)
FRR [45]	timestamps + RR	O(N)	O(1)	O(n)
Simple KPS [50]	timestamps + RR	O(n)	O(1)	O(n)
QFQ [51]	timestamps + RR	O(n)	O(1)	O(1)

There are several runtime-optimized but approximate variants of WF^2Q+ [50, 51, 54] and enhanced hybrid schedulers using time stamps and RR mechanisms [50, 51, 54] (besides the specific publications, a well-structured overview is given in [54]). The latter group makes use of aggregation of flows and a two-level architecture: a WF^2Q+-like root scheduler, which schedules on aggregates, is combined with DRR schedulers scheduling individual flows. The core of these concepts is to reduce the frequency of costly, i.e. non-$O(1)$-operations of the scheduler such as adding a queue to a priority list. Table 2.1 gives an overview on fundamental scheduling algorithms and recent runtime-optimized variants.
Unfortunately, the concept of virtual time does not allow a universal, straight forward stacking of schedulers to implement a hierarchical scheduler, which for example is necessary at BNGs downstream interfaces in focus of this thesis.

2.2.5 Hierarchical Schedulers

All discussed schedulers can enforce a desired bandwidth sharing of several or many flows on one network interface, achieving different precision and fairness and requiring different computational effort. In many use cases, especially when scheduling residential broadband access networks, it is desirable not to have packet schedulers at all nodes' outgoing interfaces, but enforcing desired policies at an ingress node for all transit traffic. This is especially interesting in tree-shaped topologies where one node feeds a subnetwork otherwise not receiving traffic. Here, a hierarchical scheduler can be deployed at the network ingress, the scheduler configuration representing the hierarchical network topology with the respective policies, see Figure 2.2. In such a hierarchical scheduler, the leaf nodes also are scheduler instances, although these nodes do not manage several flows. This is necessary since all scheduling algorithms require

state at the child nodes, e.g. the deficit in case of DRR or the timestamps for timestamp-based approaches. With hierarchical scheduling, only at one node, e.g. the BNG for wired access networks, classification and the necessary information and performance is necessary, at all other interfaces simple FIFO queues suffice. Despite this limited effort, there cannot be a decrease in service quality compared to deploying single-level packet schedulers to all outgoing interfaces at all nodes. Note that usually labels are added to the packets forwarded that reflect the classification result to allow universal configurations at AGSs and ANs that do not depend on the actual classification rules. The universal labeling is often based on VLAN tags [802.1q] and DSCPs. It must be noted that not all scheduling algorithms are suitable for hierarchical scheduling. As Bennet summarized in [38], hierarchical scheduling algorithms differ from standalone schedulers in two aspects: First, the dequeue bit rate is not known let alone constant, except for the root scheduler. Second, the queues a scheduler serves usually do not preserve the packets' order, i.e. they are no FIFO queues, except of course for the leaf queues. While many time stamp-based schedulers therefore cannot be adopted to be used as hierarchical schedulers, RR schedulers obviously work without knowing the dequeue bit rate and do not need to know the sequence of next packets for their calculations. But also some time stamp-based schedulers are capable of hierarchical scheduling, most prominently WF^2Q+.

So there are sufficiently precise, fair and efficient packet schedulers capable of hierarchical scheduling that can be chosen from dependent on the use case.

2.2.6 Rate Limiting Schedulers

Often, hierarchical schedulers also require enforcing rate limits. For instance, the main purpose of hierarchical scheduling at BNG downstream interfaces is to be able to restrict policy enforcement to a single point in the network. This is only possible, if there is no packet loss and only defined queuing at AGSs' and ANs' downstream interfaces. Since the topologies of hierarchical access networks usually have decreasing link speeds from inner to outer nodes as illustrated in Section 2.1.1, the BNG must make sure for any packet it transmits that there will be bandwidth available for this packet on all links down to the subscriber. Therefore, the BNG's packet handling must never send a packet that at some downstream link (at AGS or AN) neither can be transmitted on the link nor can be buffered at that node.

This function of only sending packets that can be transmitted on a specific link is called rate shaping. Some approaches take into account the buffer attached to the respective link, while other approaches consider the rate only and inherently assume a buffer of one packet.

Applying rate shaping at the ingress allows to reliably achieve lower worst case packet delays, see below. Rate shaping may be integrated with the scheduler, e.g. [55, 56],[57]/[58] or be implemented separately [59, 55, 60]. Traffic shaping is an old and well examined (e.g. [61, 62, 63]) technique for access control in packet switched networks. The feature which we call rate shaping appears with different names in the literature, depending on the context. Traffic shaping, policing and regulating traffic are often used in case of independent implementations. Schedulers incorporating rate shaping also use terms like maximum rate controlled scheduling.

If a scheduler incorporates rate shaping, it obviously is not work-conserving, i.e. the managed link may run empty while one or more flows have non-empty queues. Generally, there is a

trade-off between work-conserving and non-work-conserving schedulers regarding buffer needs and delays [64]. This is not applicable for hierarchical access networks, where work-conserving schedulers are no alternative because packet losses need to be controlled to comply with QoS requirements.

The foundation to separate traffic shaping from scheduling is the insight that a link with a buffer attached to its ingress can be represented by a token bucket with a fill rate σ matching the link's capacity and the bucket depth ρ matching the buffer size. A token bucket is a simple concept that can be efficiently implemented. Tokens are generated at the fixed rate σ and refill the token bucket until it is full. Incoming packets are permitted to pass to the next stage, e.g. a network interface or a scheduler, only after removing tokens equivalent to the length of the packet from the token bucket. Note that this concept is sometimes also called leaky bucket, while other work applies the term leaky bucket to a strict rate shaper. Cruz called the token bucket function in his seminal work a (σ,ρ) regulator [65, 66]. Parekh and Gallager provided extensive analysis of GPS working on such (σ,ρ)-regulated flows in [67, 68]. Most importantly, they showed that the worst-case delay of such regulated traffic traversing a multi-hop path is much more tight than the sum of single-hop worst-case delays. By this, such traffic opened the door for delay guarantees in packet switched networks. This is another major argument for implementing rate shaping already at the BNG and not distributing shaping functions in the access network.

2.2.7 Packet Scheduling of Multi-Class Traffic

In many cases, the flows in a scheduler are not of the same priority, e.g. the VoIP and BE flows of a subscriber at a BNG. There are several approaches to achieve a proper scheduling in such scenarios. Typically, the scheduling of priority traffic and BE is performed separately, but the level of separation differs between approaches. In most cases, considering priorities is implemented by priority queues, i.e. by a queue whose entries (often other queues) are sorted by priority. If strict prioritization is used, traffic shaping usually is applied to prevent BE starvation.

Basically, any scheduler can be extended to support Q sub-flows of different priority within any flow by replacing the queuing disciplines of the scheduler by a priority queue managing Q queuing disciplines. For instance, in case subscribers shall be provided with separate QoS management for VoIP, video and BE traffic, this means that instead of one per-subscriber queue three queues per subscriber are used: one for VoIP, one for video and one for BE. All non-empty queues of a subscriber are inserted into a priority queue. If a flow, i.e. a subscriber, is selected by the global scheduler to transmit a packet, the head of the priority queue is dequeued. This concept can be applied to all scheduling approaches but requires traffic shaping carried out before packets enter the scheduler.

Yet there are also specific approaches that modify an existing packet scheduler to support priorities, especially modified DRR by adding global extra queue(s) for priority traffic [69, 70, 71, 72]. Often, these approaches are just called modified DRR (mDRR) without differentiation. Such algorithms are also known to be implemented in wide spread routers, e.g. Cisco ASR9000 and 12000 series [73] and Juniper routers [74].

Assessing the impact on BE scheduling, we find that basically the priority traffic is assigned a varying amount of bandwidth prior to scheduling the BE traffic. In other words, the existence of

priority traffic means that the available bandwidth is not the constant capacity of a scheduled link but this capacity minus the varying bandwidth consumed by priority traffic.

2.2.8 Best Effort Packet Scheduling at the Edge of Access Networks

This section relates the scheduling topics presented in the last sections to the intended deployment point of the contribution of this thesis, the BE scheduling at downstream interfaces of BNGs of hierarchical access networks.

GPS-relative Delay & Fairness The recent publication of algorithms addressing the efficiency of packet schedulers show that their authors consider efficiency more important than accuracy, i.e. deviation from GPS. One reason for this priority may be the unbridled increase of packet rates: While bandwidth at Internet access links increased from 56 kbit/s modems to now several 10 Mbit/s or even some 100 Mbit/s, the maximum packet size in the Internet remained 1500 bytes, the Maximum Transmission Unit (MTU) of Ethernet, the prevailing link layer technology today. At computers, even 10 Gbit/s and 40 Gbit/s network interfaces sending packets over the Internet are operated using 1500 bytes packets (for intra-Data Center (DC) often jumbo frames are used). Note also that scheduling is not necessarily performed on single packets: Any scheduling algorithm presented above allows to dequeue a minimum number of bytes (if available in the queue). This approach can be adopted when scheduling operations become a performance bottleneck. If scheduling is still performed based on single packets, for sure a *O(N)* GPS-relative delay can be accepted, especially since N is in the range of tens or hundreds in case of BNG downstream interfaces. The limited GPS-relative delay also means that these schedulers are proportional fair. So, all presented scheduling algorithms are acceptable with regard to GPS-relative delay.

Complexity The presented algorithms have no performance problems for interface speeds currently used at BNGs and subscriber numbers served by a BNG downstream interface. A rough estimation illustrates this: A 10 Gbit/s interface fully loaded by Ethernet MTU-sized packets results in about 830.000 dequeue operations per second. In contrast, current processors are clocked with 2–3 GHz and can execute most instructions in one cycle on one core. Therefore, we may allow about 1.000 (simple) operations per dequeue event. Moreover, usually the number of served subscribers is well below 1.000, the number of active subscribers even much lower. So, even scheduling algorithms with *O(N)* complexity are feasible for BNG downstream interfaces today and in the near future provided that constants neglected in the Bachmann-Landau notation are small. So, all presented algorithms are also acceptable with regard to complexity for the targeted use case.

Hierarchical Scheduling Pure DRR schedulers, most hybrid schedulers and WF^{2}Q+ can be operated in hierarchical configurations. So again, there is a broad range of suitable schedulers to choose from.

Rate Limiting Schedulers Any packet scheduler can be combined with traffic shapers, and there are some algorithms that even incorporate maximum flow rates. So, this criterion does not limit the set of candidate schedulers, too.

Multi-Class Traffic As outlined above, multi-class scheduling can always be realized by two separate schedulers for priority traffic and BE traffic and always scheduling priority traffic if there is an eligible priority packet. There is no fundamental difference to the detailed proposals explicitly supporting multi-class traffic such as the mDRR variants. Anyway, all known schedulers of commercial BNG products are mDRR variants, see Section 2.2.7. The crucial aspect of multi-class scheduling for BE scheduling at BNGs of hierarchical access networks is the reduction in capacity available to BE traffic.
Today, usually in a wired access network the only non-BE traffic is VoIP and video traffic that is explicitly privileged by the ISP. Today, this only applies to the ISP's own services, i.e. only the VoIP service and video services bundled with the high-speed Internet access (also called "Triple Play") receive higher priority. Note also that some operators do not include VoD services but only static Television (TV) services. This applies to most of the HFC ISPs. Nevertheless, all this may change in the future. VoIP services consume very low bandwidth compared to today's access link capacities and moreover their load is constant and sessions last long. Video streaming services are less nice in terms of load since they consume much more bandwidth and today typically use a Dynamic Adaptive Streaming over HTTP (DASH)-like system (see also Section 4.5) resulting in the video being transmitted in form of periodic bursts of varying size. Therefore, the remaining bandwidth available for BE scheduling varies accordingly. Generally, if there are only few priority video streaming sessions, the relative deduction in capacity is low. If there are many priority video streaming sessions in parallel, the overall consumed bandwidth is comparably smooth, so again the changes are mostly relatively small. Nevertheless, the more subscribers are served by a BNG downstream link, the smoother is the overall priority load, so the smoother is the capacity available for BE traffic.

Conclusion

Concluding, there is a broad choice of scheduling algorithms suitable for BE scheduling at downstream interfaces of BNGs. Many of these algorithms are expected not to push today's processors to their limits. If BE traffic considered separately, the available capacity is likely to vary due to bandwidth consumed by higher priority traffic.

2.3 Congestion in the Internet

This section introduces congestion and details its role in today's Internet to provide the foundation of the next section that details CC Algorithms (CCAs).

2.3.1 Introduction to Congestion

Congestion is a central term in this thesis as for the whole congestion control and congestion management community. Nevertheless, even many publications of that field of research do not define the term. This is dangerous since the definition or understanding of the term congestion differs fundamentally between publications. In the following, we provide a definition that this document is based upon. It is aligned to the definition used in modern CC research, i.e. [75]. Older research and even some of today's publications are based on a narrower understanding of congestion.

Definition of Congestion

The term *congestion* can be defined from two perspectives. On the one hand, we define a transmit interface in a packet switched network as congested, whenever a packet on that interface cannot be sent immediately because the interface is busy sending other packets. On the other hand, the term *congestion* is linked to end-to-end rate control, most prominently TCP CC. With that respect, the term congestion refers to all signs detectable by an end host that indicate an overload of any link on the path from sender to receiver. Regarding paths, both definitions are equivalent since any packet that cannot be sent immediately results in symptoms of congestion (see below) that can be detected by an end host. Nevertheless, end hosts are only able to assess the transmission path as a whole, while sometimes it makes sense to look into congestion at a specific location, i.e. at a specific interface.

Places of Congestion

Congestion may happen at any interface that transmits packets coming from one or several packet switched interfaces of the same network node that in total have higher capacity than this interface. Despite its universality, this condition is not met by many places in today's Internet. The main reason is that core networks mostly switch large traffic aggregates based on labels, often even without touching them directly on the optical layer, i.e. without demodulating the signal or parsing packets. So, BE traffic mostly experiences congestion at the network borders, i.e. in the aggregation network on the sender's side, notably at exchange points (direct ISP peering points or Internet Exchange Points (IXPs)) and in the regional access network on the receiver's side. As detailed in Section 2.1.1, the BNG is designed to concentrate all congestion in the regional access network, so in residential access networks by design all congestion occurs in the BNG packet scheduler. Note that the bottleneck may also lie within the subscriber's network, e.g. if the end host is connected by a slow WiFi link to the home gateway which is served by a faster access link.

Symptoms of Congestion

If packets cannot be transmitted immediately, these excess packets may either be dropped or be queued for later transmission. Usually, there is some buffer assigned to any packet switched network interface that may be congested. If a packet cannot be forwarded immediately—and the buffer management allows—it will be buffered, else it is discarded. This buffer may be located directly at that interface, but may also be located remotely. With respect to our topic, remote queuing is carried out in particular at the per-subscriber queues in the BNG's hierarchical scheduler, whenever an aggregation link or, at normal load, the access link receives more packets than it can transmit and therefore is congested. Both queuing and dropping packets can usually be detected and measured by the receiver and so increased delay and packet loss are the major signs of congestion. A third category of signs of congestion is explicit signaling by the bottleneck router. For today's Internet, there is only one signaling standardized for this purpose called Explicit Congestion Notification, defined in [RFC3168]. [RFC3168] defines that the sender should react to a congestion indication by Explicit Congestion Notification (ECN) just as to a packet loss. There are several reasons why ECN never reached significant deployment [76]. For our research, ECN does make a relevant difference compared to loss, so we will cover these two cases in one and mostly just speak of loss.

So congestion leads to

- additional delay and therefore usually increased Round-Trip Time (RTT),
- maybe packet loss and
- maybe ECN-marked packets.

It is important that these congestion symptoms form a kind of continuous range from light to heavy congestion: It ranges from very little delay, over delays of up to several RTTs to packet losses. Moreover, the packet loss rate may be used to assess severity of congestion in more detail.

Detection and Measurement of Congestion

The term Congestion Control refers to mechanisms which aim to control or limit congestion in the network and to avoid congestion collapse. A vital foundation for the success of CC and the diversity of CCs is that for longer flows both packet drops and packet buffering, i.e. the resulting increase in delay, can be detected and measured by a flow's receiver. Note that some systems measure RTT while others measure One-Way Delay (OWD). Unfortunately, both packet drops and delay variation, may also be caused by other reasons, e.g. packet loss by bit errors or additional delay by retransmissions on link layer or route changes.

While ECN marks are a reliable symptom of congestion, ECN usage is negligible in today's Internet (well below 1% of TCP sources [76]) for several reasons. First, although many endpoints are ECN-capable, ECN-enabled endpoints are still a minority. Second, ECN-enabled routers are rare. Third, ECN usage with TCP requires active negotiation by the client during connection setup. This rarely happens today, but this could be changed rapidly if major operating system vendors such as Microsoft and Google decided to change this behavior by an update. Finally, the ECN signals are quite frequently mangled by middle boxes in the Internet [76], rendering ECN

signaling unreliable. Recently, there have been proposals on how to deal with packet-modifying middle-boxes [77] but there is no deployment yet. All in all, ECN up to now is negligible as congestion indication. Anyway, per its standardization in [RFC3168] the semantic of an ECN-marked packet equals a packet loss, so in terms of rate calculations by the CCAs ECN makes no significant difference. Of course, ECN allows to save retransmissions, but these are rare for today's typical Bandwidth Delay Products (BDPs) anyway.

Note that the congestion level on a path or of an interface is not only of interest to the sender but also to other nodes in a network. The simplest example are AQM algorithms that aim to signal congestion appropriately so that certain goals are met. Other examples are Congestion Policing and Congestion Policing Queues (CPQs), see Sections 3.5.5 and 3.5.6.

Both loss and ECN only provide a binary signal with every packet: congestion or no congestion. This information is sufficient for many purposes, e.g. Additive Increase, Multiplicative Decrease (AIMD) CCAs (see Section 2.4.3). But for other purposes, neither such binary information nor just the instantaneous state provides appropriate information on the current congestion level. Often it is desirable to also measure (and react upon) lower levels of congestion, i.e. to measure the queue size or the corresponding delay increase. Here it is important to consider that light congestion, i.e. a small queue, may occur at multiplexing packet switched interfaces frequently even if there is no medium-term overload and the average arrival rate is less or equal to the interface's capacity. This most importantly implies that end hosts should not necessarily react to such short-term queues by substantially reducing their sending rate. Therefore, many mechanisms internally use smoothed values to characterize the relevant congestion level of an interface or path, such as RED uses an Exponentially Weighted Moving Average (EWMA) of the queue length. Moreover, it is important that all CCAs control congestion but to do so they at intervals provoke congestion, depending on the algorithm more or less aggressively and more or less frequently (see Section 2.4).

The Term Congestion in other Work

Other publications, even publications in CC research, use different definitions of congestion. Often, the term congestion is limited to loss and ECN-Congestion Encountered (ECN-CE) marks, e.g. [RFC7713], which makes sense when focusing on BE CCs which make use of just these two signs of congestion, see Section 2.4.2 for details. In some contexts, the term congestion is even used to describe a medium-term network state with repeatedly higher bandwidth demand than capacity available, e.g. [RFC6057], see Section 3.5.3, which we would call peak load. For this thesis however, such restrictive definitions do not make sense. We use congestion as defined above: As term for any information on overload a receiver may possibly gather, in particular including delay variations.

2.3.2 General Congestion Control

CC mechanisms set up distributed systems consisting of sender and receiver(s) since the sender, which must adapt his rate to achieve these goals, cannot measure congestion experienced by his flows himself. Today, most of the Internet traffic is elastic, i.e. it adapts to available bandwidth,

typically in a huge range. The reason for this handy behavior is that for most services there is a proper reaction to congestion and their rate is adapted by at least one mechanism.

When looking at video streaming for instance, one can find rate adaption by up to four control loops:

The human audience Users will stop viewing a video when playback is repeatedly interrupted for buffering. This highest layer control loop is often forgotten, but is a non-negligible factor today, especially since most high-volume traffic is multimedia traffic directly consumed by human audience. Nevertheless, the reaction time is often in the range of seconds, although a reaction as fast as possible is desired. One reason for this slow reaction is that the user technically does not react to congestion itself but the service quality impairment induced by congestion. With more and more Machine-to-Machine (M2M) communication, this control loop may lose relevance.

Service adaptation When the video player detects imminent buffer underrun, i.e. there is a high probability that playback must be paused for buffering if no action is taken, modern scalable video services seamlessly switch to lower resolutions or lower quality versions of the same video which require less bandwidth [78, 79]. Nevertheless, such mechanism is only available for a limited set of services, typically multimedia services.

Switching service source In case of today's big video streaming providers, the streaming service is provided by multiple Content Delivery Networks (CDNs). If the achievable QoS and therefore QoE is not sufficient, the streaming client may switch from one CDN to another [80] and by that effectively relieve the network path from the old CDN to the client. Again, such option does only exist for few services and only works if the bottleneck is not also part of the new path.

Transport layer congestion control Today's ISP-independent video streaming providers use TCP (defined by [RFC793], updated by [RFC1122, RFC3168, RFC6093, RFC6528]) as transport layer protocol. One important feature of TCP is its robust rate control function, usually simply called Congestion Control (CC). Such functions have been subject to comprehensive research and many CCAs have been proposed. All relevant proposals are united in

- cautious rate increase
- fast and vigorous rate reduction in case of severe congestion

Transport layer congestion control can react to detected congestion as soon as the sender becomes aware of this information. This is up to one RTT later than the actual event. For many connections in the Internet, the actual delay is not significantly smaller than the RTT since the delay on both directions is usually dominated by the access link. So, reacting immediately after the congestion information reaches the sender is the fastest possible control loop and often is much faster than possible by the other mechanisms listed above.

Summarizing, rate control may be executed by the transport layer as well as at the service and the user level, if applicable. Amongst these, the transport layer provides the fastest reaction and the broadest availability, and works for all types of services and for M2M communication. Therefore, transport layer congestion control is a crucial mainstay of today's Internet's stability.

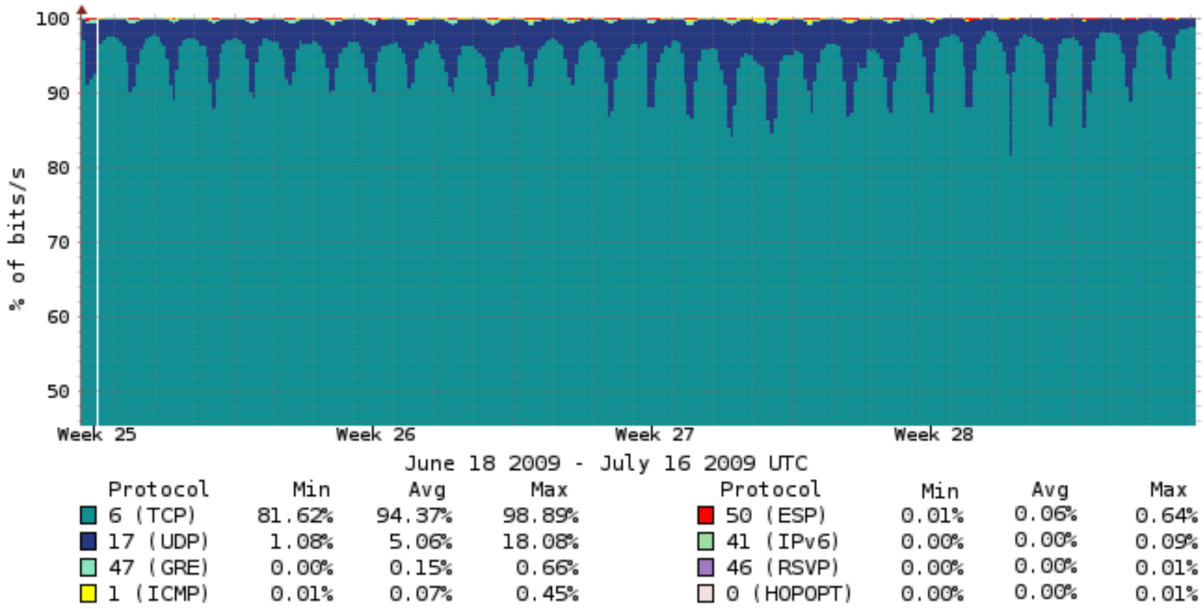


Figure 2.6: Traffic shares by protocol during four weeks in 2009 (from [81])

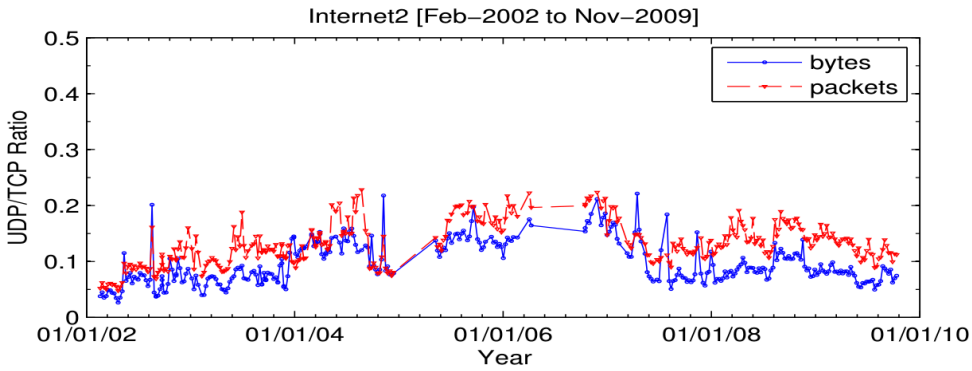
2.3.3 Prevalence of Protocol-based Congestion Control in the Internet

Almost all traffic in today's Internet is subject to automated load-adapting end host CC implemented by standardized protocols on transport but also on application layer, the most important and most prominent protocol being TCP. From the network operator's perspective, the share of adaptive and non-adaptive traffic is crucial because only the adaptive traffic prevents the network from congestion collapse. In this section, we will first discuss the prevalence of CC for unicast but will also summarize the situation for multicast.

Unicast

When assessing published traffic measurements, it is crucial to consider the chosen method of differentiation and at which networks or locations the measurements have been taken during which time. There are publications working solely on protocol fields, e.g. the IP protocol field, which is reliable information. Others derive from such information basic service types, e.g. map TCP port 80 to web traffic. Such approaches are obliged to show some false positives and false negatives. Other classification is based on DPI, which is of course error-prone but the error is hard to impossible to estimate. Obviously, generally holds that the more specific is the deducted information, the higher is the uncertainty.

Per published measurements, about 70–95 % of the traffic volume in the Internet is TCP traffic [81], which is rate controlled by design of the protocol. The remaining traffic is mostly User Datagram Protocol (UDP) traffic [81, 82]. See also Figure 2.6 taken from [81], showing traffic shares by layer 4 protocol, i.e. the IP header field "protocol". The available numbers vary over time and depend on location and ISP. See for instance Figure 2.7 taken from [81], showing the evolution of the UDP to TCP ratio over several years. It shows an increase in UDP traffic from early 2002 to mid-2004 and a decrease from late 2006 to mid-2007. The authors attribute these



83

Figure 2.7: UDP to TCP ratio during the years 2002 to 2010 (from [81])

changes to local reasons at the measured network. Nevertheless, also this figure shows that TCP is the by far predominant protocol in terms of volume.

But even packets that do not carry a TCP header on top of their outermost IP header are often subject to higher-layer rate control. Relevant examples in terms of bandwidth share are even subject to TCP's rate control, namely packets of tunneled network connections, mostly Generic Routing Encapsulation (GRE) tunnels or Virtual Private Networks (VPNs). Both types of flows carry traffic aggregates that are like the overall traffic, i.e. mostly TCP traffic. Therefore, also the rate of such aggregate flows is regulated by TCP's rate control algorithms. The volume of tunneled traffic in private residential Internet access networks increased in the last years, partly due to people working at home but using their company's infrastructure remotely, partly due to people using VPNs to sidestep geoblocking implemented based on IP addresses. But using UDP as transport layer protocol does also not mean that there is no rate control in charge, just that there is no rate control implemented by the Operating System (OS). The two in terms of volume biggest traffic groups among UDP traffic implement rate control on application layer: BitTorrent and related P2P traffic and Real-time Transport Protocol (RTP)-based multimedia streaming.

In 2013, BitTorrent P2P traffic was probably accountable for about 20% (10 % safe classification, 10 % by heuristic) of traffic volume on a large European IXP [82], but the ratio varies heavily from country to country. More recent data show BitTorrent shares between 2 % and 18 % of the overall downstream volume [83, 84, 85], see also Section 2.4.3.1 and Figure 2.10 for details. BitTorrent and derived P2P applications implement an own rate control protocol called uTorrent Transport Protocol (uTP) [86]. Before February 2010, BitTorrent had used TCP. The switch to uTP caused a significant increase in UDP traffic in several ISPs' residential broadband traffic [13]. The uTP protocol is widely used and today accounts in some regions for big amounts of traffic, e.g. Sandvine reports 58% of wired access upstream peak period traffic in Asia-Pacific for 2015 [83] and much less, but still 23% of downstream. Nevertheless, BitTorrent and P2P traffic is decreasing in the last years in many areas of the world, e.g. it only makes up 2.7% of wired access downstream traffic in North America at the same time and measured with the same methodology [87].

Then there is multimedia streaming traffic using RTP (defined by [RFC3550], since then updated in minor aspects) and the RTP protocol family. RTP is usually run on top of UDP and adds, among other capabilities, the capability for reactive rate control with RTP Control Protocol (RTCP) providing receiver feedback to the sender. There have been efforts to implement RTP flows

supporting TCP-like behavior [88]. Recently, the Internet Engineering Task Force (IETF) working group RTP Media Congestion Avoidance Techniques (rmcat) [89] brought forward new proposals for RTCP signaling [90, 91] as well as for a CCA [92]. So, also RTP / RTCP multimedia streaming can be expected to react to heavy or permanent congestion by adapting its sending rate to the network's current capacity. Nevertheless, the existence of the IETF working group shows, that RTP congestion avoidance is still a topic in research and standardization. Nevertheless, inter-domain RTP-based video streaming is rarely used. Today's well-known video streaming services, such as Netflix, Youtube, Amazon Video etc., use DASH or DASH-like technology, i.e. TCP as adaptive transport layer protocol.

There are two groups of traffic relevant for wired access networks that typically do not adapt their rates: non-adaptive multicast UDP video streaming and most types of VoIP traffic. Notably, non-adaptive UDP-based video streaming is close to non-existent in the BE traffic class. Regarding VoIP traffic, there are three important properties: First, VoIP traffic is not a lot of bandwidth in networks serving residential private customers. Mind that the traffic belonging to the ISP's VoIP service usually receives priority QoS, so it is not BE traffic. So there just remain services such as Sipgate, Skype and Viber, which are not widely used since the plans of today's ISPs usually include telephony without extra cost. Second, many VoIP services base on Session Initiation Protocol (SIP) and RTP, so service degradation could be signaled. Nevertheless, when detecting congestion there is hardly another option than canceling the call for most VoIP transmissions since there often is no option to reduce the sending rate. But, third and most importantly: VoIP service is consumed by human audience. A human user usually will cancel the call even at rather low loss rates since perceived quality of telephony is heavily degraded if any packet loss occurs.

Summarizing, in today's networks and especially in residential broadband access networks there is only negligible unicast traffic volume present that is not rate-adapted. By far most of the traffic is TCP, in most networks followed by UDP traffic that is rate-controlled by uTP or similar approaches.

Multicast, Broadcast and Anycast

IP defines more addressing schemes than unicast, namely also multicast, broadcast and anycast, which we cover in this section. Regarding UDP-based video streaming, most such services use RTP along with RTCP for control. RTP / RTCP provide support for feedback from multicast receivers [RFC5760, RFC6128] and there is a standard [RFC4654] for using this feedback for CC of the multicast transmission. In that setup, feedback from any single receiver is received less frequently and the sender shall adapt to the smallest capacity available at his receivers. Anyway, this results in serving a large set of clients with less than possible bandwidth and by that, quality. Therefore, such systems are rarely found in BE traffic. Some ISPs, which also offer video streaming services as part of their triple play plans, use RTP multicast for distribution of live TV programs, e.g. Deutsche Telekom, Hansenet and Arcor in Germany [93]. These services usually are prioritized, i.e. their traffic is not in the BE class.
Moreover, multicast within BE Internet traffic is usually not exchanged between ISPs. An important reason for this is complexity of multicast accounting: Unicast transit and peering contracts between ISPs can simply base on bandwidth or volume, since the receiving ISP's cost mostly scales with incoming volume and this metric can be easily measured by both parties.

In contrast, multicast traffic may or may not be duplicated many times in the receiving ISP's network, so the cost of incoming traffic cannot be easily estimated but would require detailed monitoring and accounting.

For broadcast transmissions, congestion control often does not make sense and is not applied although concepts for multicast CC could be adopted. Broadcast transmissions address only the hosts sharing one physical broadcast link, e.g. a satellite link, or a limited domain such as an Ethernet broadcast domain. Today, such transmissions are either unidirectional, i.e. there is just one sender and an arbitrary number of receivers as used in several wireless technologies, or broadcast is only used for few and short transmissions such as Address Resolution Protocol (ARP) [RFC826] in IPv4 Ethernet domains. Note that broadcast transmissions are increasingly replaced by multicast transmissions, see for instance the IPv6 Neighbor Discovery Protocol (NDP) [RFC4861] compared to ARP [RFC826].

Anycast traffic from the transport layer's perspective effectively equals unicast traffic. Anyway, anycast addressing is rarely used for stateful communication such as any connection-based communication which is the basis for a feedback-based CC.

Summarizing, today there is close to no non-unicast traffic in public BE traffic, also in broadband access networks. Therefore, we will focus on CCs for unicast traffic in the following.

2.4 Transport Layer Congestion Control

In this section, we give an overview on transport layer CC Algorithm (CCA). Specifically, we explain general principles, how prioritization can be achieved by CCAs and which requirements need to be met. Moreover, we introduce selected CCAs that are most relevant in today's Internet and present their classification into foreground, i.e. BE, CCs and background, or Lower than Best Effort (LBE), CCs.

If reliable transmission is implemented by retransmitting lost packets without proper rate control, congestion may lead to a stable state [RFC896] where a huge fraction of the packets transmitted are unnecessary retransmissions. In the 1980's that happened frequently in the Internet. These incidents were called *congestion collapse* [RFC896]. This state is well beyond the *knee* load, referring to Chiu and Jain's state classification in their seminal publication on congestion avoidance algorithms [94].

From this, the protocol designers learned that congestion is the consequence of the network being not in equilibrium, i.e. the number of packets in the network changes [95]. Nagle derived "packet conservation" as the goal of congestion control.

While congestion collapse for sure must be prevented, congestion practically cannot be avoided in packet switched networks if only because the capacity of a path is not known in advance and typically changes over time. So for an end host, the only way to detect the current capacity of a path in the Internet is to drive resources to their limit, but this obviously means generating congestion carefully and only up to a certain extent. This is what happens every moment in

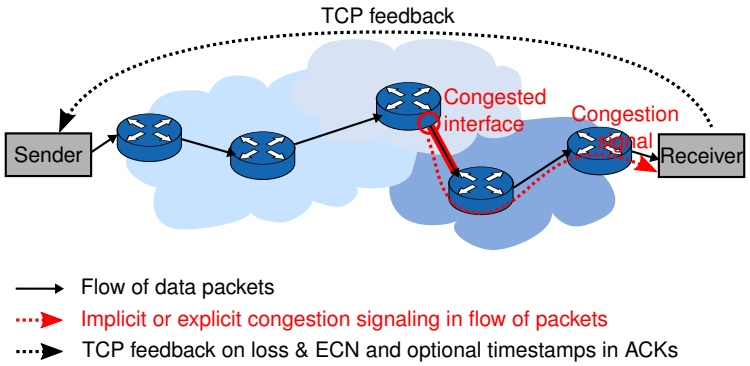


Figure 2.8: Flow of information in the TCP control loop (symbolic figure)

innumerable hosts in today's Internet, which obviously does not suffer from a congestion collapse. So, in general congestion only is evil if it is not dealt with properly.

Controlling or managing congestion always comes with the challenge that the receiver of a transmission is the end host able to detect congestion, but the sender needs to act, i.e. re-act to the latest congestion measurements. Luckily, most transmissions are desired to work reliably, so there is some bidirectional communication set up between the communicating end hosts. This obviously is the case for TCP, the most wide-spread transport layer protocol.

Nevertheless, there are more protocols providing rate control that can be seen as transport layer congestion control since these protocols are based on UDP that itself does not provide such functionality. The most prominent and most relevant example is the already mentioned uTP used by BitTorrent. uTP is sometimes also referred to as "micro Transport Protocol", highlighting even more that this protocol may be seen as transport protocol or part of a transport protocol. uTP provides functions that the underlying UDP does not provide: it provides connection management, reliable transmission, ordered delivery and, most important from our perspective, a rate control implementing CC.

2.4.1 General Principles of Transport Layer Congestion Control Algorithms

Every CCA has several partly conflicting goals and the balance among common goals differs from algorithm to algorithm. Nevertheless, all CCAs share the goals of on the one hand successfully transmitting data from sender to receiver and on the other hand controlling the congestion created in the attempt to do so.

Figure 2.8 shows the flow of information in the control loop of a TCP connection. The sender sends packets according to its CCA. These packets may suffer from congestion anywhere on the path. This congestion is perceived by the receiver, which signals back this information to the sender, which then reacts appropriately according to its CCA based on the updated state. Obviously, there is always one effective RTT between a change in congestion at a certain bottleneck and the reaction of the sender becoming effective at this very place. This feedback delay in the closed control loop consists not only of the transmission delays of the links on the path but also of the queuing delays at all interfaces on the path.

A good CCA should be scalable, i.e. should perform well according to its specific goals for a wide range of link speeds and RTTs in arbitrary combinations. Moreover, a CCA should also be robust regarding network behavior, especially regarding packet reordering and packet loss not caused by congestion but for instance bit errors. Unfortunately, many TCP CCAs, including wide-spread ones like TCP Cubic (see Section 2.4.3.3), do not work well in presence of non-congestion packet losses. This fact and that wireless transmissions do have non-negligible bit error rates when using high efficiency modulations, lead to an interesting—some people say weird—development: Since the wide-spread TCP CCAs do not cope well with losses but ignore delay variations, most wireless standards such as WiFi [96] or cellular networks standardized by the 3rd Generation Partnership Project (3GPP) use link-local retransmissions. It underlines the outstanding importance of TCP's CCs in today's Internet that these mechanisms violate the end-to-end principle [97] the Internet is supposed to be based on.

A CCA generally aims to reliably transmit a stream of data to a counterpart instance on another host over a potentially error-prone packet switched channel of unknown and time variant capacity. It therefore has several tasks.

1. At startup, it has to acquire bandwidth appropriately fast.
2. During transmission and if there is no congestion, it must probe appropriately for newly available bandwidth.
3. If there is congestion, it must react appropriately.

Obviously, there are two phases that can be distinguished, startup and ongoing transmission, and there is a lot of freedom in defining what the word *appropriately* leaves open. We will provide an overview on potential choices and rationales, and introduce core technical terms.

All relevant CCAs are based on the sender managing a so-called Congestion Window (cwnd), a specific type of sliding window that can grow and shrink. Focusing on the CCA and thus leaving aside receiver limitations, this concept describes a sending algorithm that maintains three pointers in the buffer of data to be transmitted:

1. A pointer to the newest unit that the CCA allows to transmit, i.e. the upper limit of the cwnd.
2. A pointer to the oldest unit that has not been acknowledged by the counterpart instance, i.e. the lower limit of the cwnd.
3. A pointer to the newest unit that has been transmitted, usually within that limits.

For the startup-phase, the different types of CCAs have different goals:
Foreground CCAs aim to grab a significant bandwidth share reasonably fast. Moreover, such algorithm accepts causing congestion in that phase, causing potential other flows to step back and to free up bandwidth, allowing to reach an appropriately fair allocation fast. As of now, all foreground CCAs implement the same behavior during start-up. During this so-called *slow start*, the receiver sends one Acknowledgment (ACK) for each received packet, i.e. the delayed-ACKs mechanism defined in [RFC1122] is disabled, and the sender sends out two packets for each received ACK. By this the cwnd is doubled every RTT, resulting in an exponential growth of the sending rate while there is no queue and the RTT is constant.
Background CCAs in contrast aim to not disturb potential foreground traffic, so they start less aggressive, and aim to detect resource shortage before foreground flows react to their presence. The CCA of uTP for instance increases linearly by adding two segments per RTT to the cwnd.

Therefore, such flow will take a long time to utilize a path with a big BDP, but the impact on the congestion level, that is caused by such sender but not yet detected by it, is small.

Standard TCP uses a state variable called slow start threshold (ssthresh) to decide on when to leave slow start and enter the so-called *congestion avoidance* phase. The slow start phase is also ended when the CCA detects congestion (based on specific algorithm's detection algorithm). The slow start algorithm is used when cwnd < ssthresh, while the congestion avoidance algorithm is used when cwnd > ssthresh [RFC5681]. The following phase is controlled by a different part of the CCA, the congestion avoidance algorithm. Congestion avoidance is where foreground CCAs differ but this is also the part of the CCA that has the biggest impact on important metrics, e.g. the throughput that can be achieved in certain scenarios or the bandwidth sharing with other flows. Any congestion avoidance algorithm basically serves two purposes:

1. to probe for higher bandwidth by increasing the sending rate if no congestion is detected
2. to reduce congestion by reducing the sending rate if congestion is detected

For loss-controlled CCA, which base on a binary feedback, potential congestion avoidance algorithms can be roughly classified by the class of function they use to increase and decrease the cwnd in these two cases. If then a binary function $c(t)$ represents having detected no congestion ("0") or congestion ("1") since packet transmission t, then all possible linear functions to adopt the cwnd follow the formula shown in Equation 2.3.

$$cwnd(t+1) = \begin{cases} \alpha_i + \beta_i \cdot cwnd(t) & \text{if } c(t) = 0 \\ \alpha_d + \beta_d \cdot cwnd(t) & \text{if } c(t) = 1 \end{cases} \tag{2.3}$$

Depending on which parameters in this notation are not neutral, the CC functions can be classified into four classes:

- Additive Increase, Additive Decrease (AIAD),
  i.e. $\alpha_i > 0;\ \alpha_d < 0;\ \beta_i = 1;\ \beta_d = 1$
- Additive Increase, Multiplicative Decrease (AIMD),
  i.e. $\alpha_i > 0;\ \alpha_d = 0;\ \beta_i = 1;\ 0 < \beta_d < 1$
- Multiplicative Increase, Additive Decrease (MIAD),
  i.e. $\alpha_i = 0;\ \alpha_d < 0;\ \beta_i > 1;\ \beta_d = 1$
- Multiplicative Increase, Multiplicative Decrease (MIMD),
  i.e. $\alpha_i = 0;\ \alpha_d = 0;\ \beta_i > 1;\ 0 < \beta_d < 1$

The basic goal of a congestion avoidance algorithm is to achieve both efficiency and fairness at the same time, i.e. in the long run rates or at least volumes of competing flows should converge. Fundamental research [94] on these algorithm classes showed that both AIAD and MIMD do not change a resource allocation over time in any direction, neither to a fairer nor to a less fair allocation. MIAD algorithms even increase unfairness and result in starving flows. So, all successful congestion avoidance algorithms based on binary feedback use the AIMD principle. Some use a straight forward implementation, but for big BDPs strict linear growth results in very slow acquisition of bandwidth. Therefore, many congestion avoidance algorithms have been proposed that grow faster than linear. We describe different approaches for congestion avoidance of selected relevant CCAs in Section 2.4.3.

Here we want to give some remarks that are independent of the actual CCA:

Unit of Operation A CCA's unit of operation may be bytes as well as packets. Operation in bytes, also known as Appropriate Byte Count (ABC), has been standardized in [RFC3465] and there used to exist a Linux implementation. This implementation has been removed already in 2013 [98] and we know of no other byte-based implementation. So all relevant CCAs and all CCAs presented hereafter operate on packets.

Unavailability of Data In real implementations the amount of available data must be managed too, typically using a fourth pointer. The upper limit may grow beyond that pointer, i.e. some data between the current pointer and the upper limit may not be available yet. If the current pointer reaches the limit of available data, sending is interrupted until data becomes available. Two cases can be distinguished, which both result in loss of self-clocking.
Case 1: No data is available for a duration longer than the Retransmission TimeOut (RTO) (which is at least one second [RFC6298]). In that case, the cwnd shall be reduced to its initial size of ten packets as standardized in [RFC2581]. This usually is less than the ssthresh, so slow start will be carried out, inducing the typical overshoot and its large packet loss of about one BDP.
Case 2: The delay is less than the RTO. Then the intermission in sending packets inherently means that the self-clocking mechanism cannot be used for the respective range. So, it is crucial how fast data are sent, once they become available. If the data are sent in one burst, self-clocking is thwarted also for the near future. Another problem is that the bottleneck is often much slower than the sender's interface speed, so if the sender sends the data—up to a full cwnd—at interface speed, many packets are dropped at the bottleneck's buffer. Pacing, i.e. inserting packets into the network in intervals, e.g. of $\frac{RTT}{cwnd}$, supports smooth functioning of the self-clocking despite the intermission.
This topic is important for bursty traffic, e.g. it is a known issue for VoD traffic [99]. Although there have been several proposals to limit the rate of packets entering the network in TCP, e.g. [100], it seems the better solution to make use of a fair queuing discipline, effectively achieving the same goal [101].

2.4.2 Bandwidth Allocation: Fairness Challenge and Opportunity for Prioritization

Since the CCA determines the sending rate of an end host, it has a huge impact on bandwidth allocation. This has many interesting aspects: On the one hand, this poses security and fairness concerns, but on the other hand this also provides an opportunity for end hosts to affect priorities in bandwidth allocation without explicit signaling. We will discuss both aspects.

2.4.2.1 Fairness in Bandwidth Allocation by CCAs

When it comes to fairness, the granularity of considered entities is fundamental: Fairness can be examined between subscribers, between hosts (of the same or different subscribers), between services or between flows, e.g. TCP connections. When talking about CC and fairness, it is always about fairness among flows. Nevertheless, in many cases this is not the relevant metric from the user's perspective. But this is the level a CCA works on and moreover, fairness and

unfairness on this level are the foundation for higher level fairness and therefore fairness on this level has substantial impact on such metrics. Therefore, some consensus established in the networking, particularly in the transport layer community, about how fair a flow should behave when competing with other flows for resources, i.e. bandwidth and buffer space.

This minimum fairness expected from a CCA is often termed *TCP friendliness* and referred to even in standards [RFC5166, RFC5348]. TCP friendliness requires a CCA to not acquire on average significantly more bandwidth than standard TCP does in the same situation. This is equivalent to one flow of the examined CCA sharing a bottleneck about fairly with one standard TCP flow if competing at a bottleneck. So although everybody talks about fairness, this requirement is more about an upper limit for aggressiveness in seizing bandwidth rather than a target behavior. Note also that the basic goal behind TCP friendliness is not unquestioned in the research community, see e.g. [102, 103]. But there are even more difficulties regarding fairness in bandwidth allocation by CCAs. Many CCAs are not necessarily fair even among flows of the same CCA if the circumstances differ. Such difference may be different RTTs or the queue size at a flow's start time. Regarding fairness, RTT-unfairness is the most important issue in the Internet today. The core reason is that the actions of CCAs depend on the congestion experienced by their lately transmitted packets but they can only learn about that one RTT later. Nevertheless, available and deployed CCAs differ in the impact of the RTT on their operation.

In many publications [104, 105] TCP, i.e. Reno or NewReno, is also said to be convergent, i.e. that the rates of flows of the same type would get closer and closer until an equilibrium is reached. This indeed happens if all flows participate in every congestion event (definition given in Section 2.4.3.2), i.e. in case that if one flow experiences a loss, all competing flows also receive a loss within one RTT. This results in all flows synchronously increasing and decreasing their rates, that's why this state is called *global synchronization*. Global synchronization usually does not occur in reality due to the packets being sent in bursts over the bottleneck that is faster than the sending rate [106]. The initial window of ten packets (see [RFC6928]) as well as the pairs of packets caused by using delayed acknowledgments (see [RFC1122]) are sent back-to-back. The bottleneck spaces these packets, i.e. it induces self-clocking (also called ACK-clocking). But this self-clocking is aligned to the actual bottleneck speed, and neither to the current sending rate nor to the fair share aimed for. So, the packets of the cwnd of a flow do not get evenly spaced over the RTT, but remain partly in bottleneck-speed bursts. These bursts result in not all flows being hit by a packet loss when the buffer reaches its limit, but just a random subset. In consequence, a flow's rate deviates significantly from its fair share (the rate's expectation value) for extended periods of time that are longer than the time between congestion events (called congestion epoch, see Section 2.4.3.2 for a detailed definition). For standard TCP, TCP NewReno, a flow's rate deviates at least by a factor of three.
Nevertheless, for competing flows of the same type the overall transmitted volume is normally distributed and the variance approaches zero for the time approaching infinity due to the central limit theorem. For details on that topic see Lautenschläger's analysis and experimental evaluation in [106].
So, if several flows with same conditions and controlled by a loss-controlled CCA share a bottleneck there is fairness among the flows in the long-term average, but there usually is significant unfairness in short and medium terms. This is an important argument to use schedulers rather than shared queues where possible.

2.4.2.2 Opportunities for Prioritization by the End Host

The picture of fairness gets more diverse and complicated when not only binary feedback, i.e. not only loss and ECN, is considered. Soon after Jacobson identified the required properties to be met by CCAs to avoid congestion collapse and proposed the initial Tahoe CCA [95], researchers aimed at developing CCAs that do not induce that many packet losses as Tahoe. For this purpose, delay-controlled CCAs like DUAL [107] have been invented that react already on increasing delay and do not wait until packet loss is detected. Nevertheless, packet switched interfaces are usually equipped with significant buffers to cope with small peaks in load, e.g. as caused by several packets arriving at different input links of a router or switch at the same time. Delay-controlled CCAs detect an increasing queue by the caused increase in delay and reduce their cwnds substantially earlier than loss-controlled CCAs, which instead further increase their rate until packet loss is detected. Therefore, loss-controlled flows competing with delay-controlled flows for a bottleneck effectively results in unfair bandwidth allocation up to starvation of the delay-controlled flows. Since TCP's standard [RFC5681, RFC6582] CCA NewReno requires one BDP of buffer at the bottleneck to allow a single flow fully utilizing the bottleneck's capacity (see Section 2.4.4), buffers are usually sized rather big, i.e. according to that requirement applied to some pessimistic RTT estimation. Because of these large buffers, the effect described above typically strikes hard in today's networks, i.e. delay-controlled CCAs receive much less capacity when competing with standard TCP flows. In contrast, delay-controlled flows typically react fast enough so that a starting loss-controlled flow is not significantly impacted even if delay-controlled flows fully utilize a shared bottleneck at its start time. In consequence of the described interaction between foreground and background traffic, foreground traffic dominates not only the bandwidth allocation, but also the behavior regarding buffer allocation: As soon as there is one sufficiently large foreground transfer, the queue periodically grows until at least one packet is dropped and then shrinks fast only to repeat the growth phase. Due to standard TCP's slow start behavior, rather little volume, few BDPs, is required for a foreground transmission to achieve this effect.

This relationship can be used by end hosts to implement prioritization by using a loss-controlled CCA for normal and more urgent traffic and a delay-controlled CCA for non-urgent traffic (for a good overview we recommend the survey by Ros and Welzl [75]). From the user's perspective, this represents a differentiation between on the one hand foreground traffic and on the other hand background traffic that is only transported if resources suffice after serving all foreground needs. We therefore also use the terms foreground and background with respect to CC, CCA and flows.

From the ISP's perspective, all the traffic, foreground and background, belongs to the BE class. Since background flows nevertheless receive worse service than could be expected from BE, the term LBE is used in the literature, too [75]. Protocols using background CCAs are also termed scavenger protocols in the literature ([108, 109, 110]), or low priority CC ([111], republished as [112], [113]).

Such traffic prioritization by CCA is effective between all flows passing the same bottleneck, especially such control is not restricted to work among the flows of one host. This may be a beneficial property as well as severe drawback, depending on the location of the bottleneck.
On the one hand, for the residential subscribers in focus of this thesis the bottleneck in most cases is their own access link which results in a very desirable system behavior: Within the subscriber's

household, background traffic fills bandwidth left by foreground traffic and foreground traffic does not receive significantly less bandwidth than without that background traffic. More important, this means that the users' QoE is not impaired by the background transmissions taking place. For example, if a subscriber's device A, e.g. the desktop computer, receives huge downloads controlled by a background CCA, e.g. software updates, this traffic will make room for foreground traffic, e.g. the VoD streaming traffic for a TV set-top box. If both transmissions use standard TCP instead, i.e. the update transmissions also use a foreground CCA, the resulting bandwidth allocation might leave not the bandwidth necessary for the VoD streaming. This in turn results in either stalls or reduced quality, depending on the used technology, but in any case, in a QoE deterioration. So, if the bottleneck is the access link, the subscriber receives direct benefit.
On the other hand, if the bottleneck is not the access link, e.g. at a peering point or at an aggregation link, the background transfers make room for any other foreground traffic, i.e. there is no immediate advantage for the subscriber. If congestion is sustained, e.g. due to a poorly dimensioned peering point, the background transfer receives low bandwidth over a long time. This may at some time result in dissatisfaction even for non-urgent background transfers. This momentary drawback could be compensated if the subscriber would benefit from other users' background traffic yielding to his foreground traffic at that bottleneck in the future. The probability of yielding paying off depends on where the bottleneck is. While it is high for bottlenecks in the regional access network, it is probably lower for more distant bottlenecks at peering points. Fortunately, bottlenecks outside of the access network occur rarely, so Apple decided to use a background CCA for its update distribution [114]. Also the OS designers at Microsoft implement a background transmission technology for their software updates [115] but there is no information on how Microsoft achieves that goal, probably not by just using a background CC. Keep in mind that the actual deployment of the background CCA is needed at the sender's side, not at the subscriber's host.

The relation between a traffic's priority or QoE relevance and its CCA is asymmetric: If traffic is controlled by a background CCA, it is of low priority at this time. That means it does not matter if this traffic is transported, i.e. it does not impact QoE for three reasons: First, using a background CCA may result in close to zero throughput when there is competing foreground traffic. Second, there are many transfers that have no importance until some point in time, e.g. the software update that is only needed until the next reboot. Third, importantly, if the resulting throughput rate is not satisfactory, the sender can switch to a foreground CCA at any time. If traffic is controlled by a foreground CCA, it may be QoE relevant or it may not. Background traffic may be using a foreground CCA for many reasons, e.g. since the service provider did not know about traffic differentiation by CC and used the standard reliable transport layer protocol.

Summarizing, a delay-controlled CCA can achieve lower packet delay and jitter as well as low loss rates if there is no competing loss-controlled traffic. If delay-controlled traffic competes with loss-controlled traffic, it will receive only a small capacity share, so the choice of the CCA can be used to implement end host prioritization. Moreover, as soon as there is a single, sufficiently large foreground connection in a bottleneck aggregate, it pushes back any existing background connections and dominates the bandwidth and buffer allocation at the bottleneck. Lastly, traffic controlled by a background CCA does not contribute to QoE.

2.4.3 Selected Congestion Control Algorithms

Over the last 35 years, since the initial standardization of TCP [RFC793], many algorithm proposals have been published for TCP's core CCA. The vast diversity of CCAs has been developed for very different motivations. That proves that the performance of the existing algorithms did not satisfy the requirements of all scenarios or users at that time. Despite the huge number of published algorithms, there are only few that gained significant relevance in the public Internet. While use in the public Internet was the declared goal for many proposals, some CCAs also have never been intended to be used in the public but have been designed for certain, separated domains. Examples are TCP Hybla [116] intended for use on links with large RTTs, notably satellite links, and Data Center TCP (DCTCP) [117, 118] intended for use in DCs with a special AQM configuration at all switches.

For an overview on TCP CCA proposals, we commend the survey of Afanasyev et al. [105]. To show the fundamental groups among CCAs, we include a figure from it, Figure 2.9. It shows that the two fundamental approaches to detect congestion, loss-controlled and delay-controlled, have both been proposed very early after the congestion collapse experiences in the 1980s: the early representatives Tahoe [95] and DUAL [107] emerged 1988 and 1992. Each was shortly followed by a more elaborated version, namely Reno in 1990 (described in [119] although standardized not until 1999 [RFC2581, RFC3782]) and Vegas [120] in 1995. These two algorithms are still very important as benchmarks although both are not in large use in the Internet, see the Section 2.4.3.1.

There were repeated initiatives to develop background CCAs. The most prominent representatives are TCP Nice from 2002 [113], TCP LP (TCP Low Priority) from 2003 [111] (republished in 2006 [112]) and uTP [86] (initial version from 2009, last update in 2015). Nevertheless, the only background CCA that gained importance is uTP in BitTorrent and the equivalent Low Extra Delay Background Transport (LEDBAT) at Apple. We analyze the prevalence of CCAs in today's Internet in the following section.

2.4.3.1 Prevalence of CCAs in the Internet

Recent research on prevalence of CCAs in the Internet shows, that 17–26% servers use plain AIMD CCAs and 45% use TCP Cubic and related CCAs [121] (see Section 2.4.3.3). Some 10–19% of the evaluated servers use TCP Compound as CC (see Section 2.4.3.6). These numbers do not represent the traffic share in the Internet of the respective servers, just the sheer number of hosts using it. For example, the amount of traffic sent by a video streaming service such as youtube.com can be safely assumed to be manifold that of a text-based web site, e.g. wikipedia.org, but in these figures both have same weight. Unfortunately, there are no figures available regarding traffic volume classified by the used CCA. Nevertheless, some services responsible for huge volumes, e.g. the video streaming services Youtube [122, 123] and Netflix [124], are known to use the Linux OS on their servers. So, the traffic share of TCP Cubic, the default CC of Linux, can be assumed being much higher than its share in hosts.

For TCP Compound the picture is different. TCP Compound is patent protected and only available at the Microsoft Windows OS. On the one hand, there are no big services running

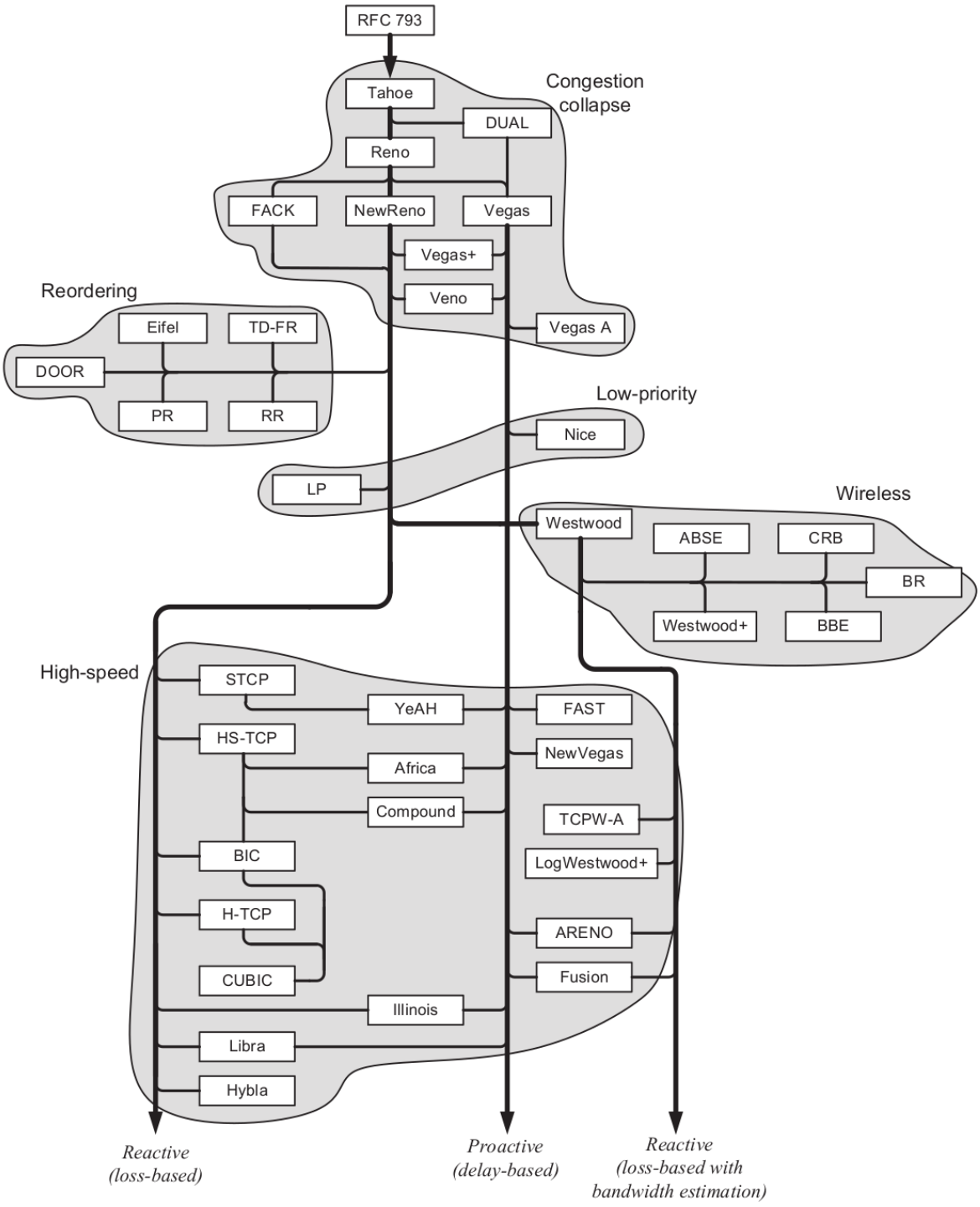


Figure 2.9: Evolutionary graph of variants of TCP congestion control (from [105])

on Windows platforms known to the author. On the other hand, Microsoft Azure supports provisioning of Windows Virtual Machines (VMs) as well as Linux VMs, all likely to cause high Internet traffic. The crucial and open question regarding this work is which role these services play and will play for residential Internet subscribers. Moreover, there is a third point: Microsoft software updates. It uses Microsoft's Background Intelligent Transfer Service (BITS) [115] which from its description does not seem to use simple TCP Compound transmissions.

Downstream	
Application	Share
YouTube	24.44%
HTTP	15.39%
Facebook	7.56%
BitTorrent	6.07%
SSL - OTHER	5.51%
Netflix	4.82%
MPEG - OTHER	3.82%
iTunes	2.24%
Flash Video	1.85%
Twitch	1.65%
	73.35%

(a) Europe, 2015 (latest available data, from [83])

Downstream	
Netflix	35.15%
YouTube	17.53%
Amazon Video	4.26%
HTTP - OTHER	4.19%
iTunes	2.91%
Hulu	2.68%
SSL - OTHER	2.53%
Xbox One Games Download	2.18%
Facebook	1.89%
BitTorrent	1.73%
	74.33%

(b) North America, 2016 (from [85])

Downstream	
Application	Share
YouTube	32.78%
BitTorrent	18.23%
HTTP (Other)	8.03%
Facebook	4.25%
RTSP (Other)	3.46%
MPEG (Other)	2.70%
iTunes	1.63%
SSL (Other)	1.52%
PC: Valve's Steam Service	1.11%
Google Market	1.10%
	74.81%

(c) Asia-Pacific, 2016 (from [84])

Figure 2.10: Top 10 peak period applications in wired access networks' downstream of different regions according to Sandvine

Besides the uncertainty about volume shares, the cited research only covers TCP traffic, neglecting all other congestion-controlled traffic, most importantly BitTorrent traffic that still accounts for a significant volume. Nevertheless, there is data available on prevalence of distinct services, for example from Sandvine Inc., a manufacturer of traffic management solutions mostly based on DPI (see also Section 3.5.4). Figure 2.10 shows data published by Sandvine Inc. on the shares of applications in fixed networks' downstream traffic during peak periods. These figures provide two important insights: There is a significant share of BitTorrent traffic and there is traffic mapped to the iTunes application.

The share of BitTorrent traffic, i.e. of uTP-controlled traffic, during peak periods varies heavily from region to region, from below 6 % in Europe, see Figure 2.10a, and 2 % in North America, see Figure 2.10b, up to 18 % in Asia-Pacific, see Figure 2.10c. This data is probably exclusively based on networks deploying Sandvine solutions that by design impact the shares of different types of traffic (see Section 3.5.4). Sandvine solutions aim to reduce traffic of applications undesired by the deploying operator, most prominently P2P traffic. So, BitTorrent might be underestimated in their figures. As already mentioned, BitTorrent P2P traffic was reported being accountable for about 20 % of traffic volume on a large European IXP in 2015 [82], while Sandvine reported for the peak hours only about 6 % share for wired access downstream and not even 2 % for mobile access downstream for the same period [83]. Such figures differ heavily not only from region to region but also from operator to operator. Among other factors, it depends if and how peak load management is carried out by the operator. Generally, the share of P2P traffic in the last years in Europe and North America is not as high as it was 2010 and before, probably due to intensified prosecution of illegal file sharing using this technology.

In the Sandvine figures, a rather new type of traffic is listed: iTunes traffic. Apple's iTunes application in terms of traffic mostly downloads rather big files, be it software updates, music downloads or movie downloads, yet there is no more information on which traffic exactly Sandvine accounts for by the type iTunes. Apple uses their implementation [125] of LEDBAT [RFC6817] for software update distribution [114]. Note that similar downloads of Windows

devices (from computers to smartphones) and Android devices are not explicitly listed and will probably be captured in the HTTP and SSL fractions. To our knowledge, Android does not make use of a special CCA for such downloads but due to Google being the only provider of this service, Google can change that at any instant if it sees benefit.
Summarizing, uTP (BitTorrent) and LEDBAT (Apple) make up a significant share of today's Internet traffic.

Therefore, we identify TCP NewReno, TCP Cubic, TCP Compound and uTP / LEDBAT as most prevalent CCAs. In the following, we will briefly present these algorithms, augmented by TCP Vegas, focusing on their behavior, i.e. their congestion detection and cwnd adaptation algorithms.

2.4.3.2 TCP NewReno

The NewReno CCA uses slow start, a linear increase of one packet per RTT in congestion avoidance and halves the cwnd when detecting congestion. It is standardized in [RFC6582] and its foundation, [RFC5681]. It is a loss-controlled algorithm and detects packet loss by either a timeout, the so-called Retransmission TimeOut (RTO), or by receiving three duplicate ACKs. These two mechanisms result in different reactions, so NewReno differentiates between rather light and rather heavy congestion.

The RTO is computed based on the smoothed RTT (calculated as EWMA), the smoothed RTT variation (also an EWMA) and a minimum of one second [RFC6298]. If a timeout occurs, ssthresh is set to halve the cwnd, cwnd is reduced to one segment, slow start phase is entered and the apparently lost packet is retransmitted. But detecting congestion by timeout is the rare case and happens mostly when there are no three packets following the lost packet (sufficiently fast), so the receiver cannot send three duplicate ACKs. Importantly, this does not only happen at the end of a connection but also if the application sends data in blocks and provides the next block of data only after the RTO expired. This for instance may happen for the burst transmissions of DASH-like VoD flows.

The second loss detection mechanism based on duplicate ACKs is part of a mechanism called *fast retransmit* since it includes immediately retransmitting the apparently lost segment without waiting for the RTO to expire. Fast retransmit is followed by the *fast recovery* phase. In this phase, NewReno assumes that there is only light congestion on the path and therefore only halves the cwnd. The cwnd is halved in contrast to set to one segment as in case of timeout, since every received ACK, being it duplicate or not, indicates that packets have successfully reached the receiver. So, the congestion is rather light and moreover, these packets left the network and can be partly replaced without increasing congestion.

NewReno, in contrast to the original Reno algorithm, moreover does not allow the cwnd to be halved more than once for packets within one cwnd, i.e. packets sent within one RTT. Depending on the situation, a congestion event often results in several packets lost before the sender can reduce his sending rate, i.e. within one RTT. Nevertheless, it was found that halving the cwnd for every lost packet, as the original Reno does, is not necessary and results in poor performance. So NewReno uses a pointer to the last segment transmitted before the loss and does not react to

any losses of segments before this pointer. This observation and the success of the described algorithm extension resulted in new terms having been coined:

- A *congestion event* means a time where losses occur which is not longer than one RTT.
- A *congestion epoch* means the time between two successive congestion events.

Obviously both terms are based on a congestion definition based on loss (or ECN).

Nevertheless, this original algorithm has disadvantageous properties which have been approached by several slight algorithmic changes. Due to the one-step cwnd reduction in original NewReno, the sender will not send out any packets for the next half RTT, and will send packets with the old frequency for the second half, thus impairing self-clocking. This is not desirable and may induce unnecessary congestion at the bottleneck. An equal pacing of packets would be more desirable. Rate-Halving [126] as well as Proportional Rate Reduction (PRR) [RFC6937] are changes to NewReno that spread the packets after a loss over the whole RTT. All these algorithm variants are usually just called NewReno. Linux even still uses the name reno, although it implements NewReno since version 3.2 (January 2012) with PRR, before that with rate halving.

A separate issue is that the sender cannot distinguish duplicate ACKs caused by unnecessary retransmissions, e.g. due to reordering, from duplicate ACKs that are caused by a packet loss. With standard TCP, the receiver just cannot provide the crucial information which packet has been missing. Therefore, the Selective Acknowledgment (SACK) extension [RFC2018] was introduced and standardized that allows the receiver to provide additional information about which missing segment(s) triggered a duplicate ACK. Along with algorithms for loss detection using SACK [RFC6675], the situation regarding ambiguous ACKs was improved but not fully solved due to the very limited amount of information a SACK option can transport.

Summarizing, the name NewReno stands for a family of CCAs that increase their cwnd by one segment per RTT during congestion avoidance and halve their cwnd on congestion events. It therefore is slow to seize bandwidth in congestion avoidance and the increase speed depends on the RTT.

2.4.3.3 TCP Cubic

Cubic [127] was designed to avoid NewReno's shortcomings, i.e. it aimed to scale well at big BDPs and to be fair when competing with flows of another RTT or the standard CCA, i.e. NewReno. TCP Cubic uses slow start, a multiplicative decrease of 0.8 according to [127] (but about 0.7 in the Linux implementation), fast retransmit and fast recovery and a TCP Cubic congestion avoidance algorithm that is calculated in time, not in RTTs. It basically enhances the Binary Increase Congestion Control (BIC) algorithm [128] by the approach of Hamilton TCP (H-TCP) [129, 130] that proposed to use the time elapsed since the last congestion event as basis instead of counting RTTs. The cwnd is computed as shown in Equation 2.4, where C is the predefined constant 0.4, t_{loss} is the time of detecting the last congestion event, β is

the multiplicative decrease factor of 0.8 or 0.7 and $cwnd_{loss}$ is the cwnd just before the last congestion event.

$$cwnd_{cubic}(t) = C\left((t - t_{loss}) - \sqrt[3]{\frac{\beta}{C} * cwnd_{loss}}\right)^3 + cwnd_{loss} \tag{2.4}$$

This growth formula achieves that growth does not depend on the RTT and the last maximum cwnd $cwnd_{loss}$ serves as reference for the growth in the next congestion epoch. While the cwnd is far from this reference, it grows fast, while it is close to the reference, it grows slowly and may even not grow at all for several RTTs. Moreover, TCP Cubic guarantees not to perform worse than NewReno since it explicitly calculates the cwnd NewReno would have at that time since the last congestion event and uses the maximum of both. Nevertheless, for RTTs and link capacities of today's networks, this mechanism does not make a difference since for such large BDPs the cubic function grows much faster than the linear one of NewReno.

Summarizing, by its RTT-independent congestion avoidance algorithm TCP Cubic provides better fairness and faster utilization of newly available bandwidth for large BDPs. In all other respects, it inherits the well-working properties of NewReno, e.g. slow start and the mechanisms for congestion detection.

2.4.3.4 TCP Vegas

TCP Vegas [120] is the most prominent delay-controlled CCA. It was developed to improve the periodic behavior of TCP DUAL and TCP Reno / NewReno that results in oscillations in sending rate, RTT, queue size and periodic packet loss. Therefore, TCP Vegas tries to estimate the queue size at the bottleneck by comparing RTT measurements with the lowest RTT seen. It then adopts the sending rate to keep the queue size in a narrow range close to no queue. The original Vegas algorithm also changes the startup phase to allow RTT increases to be detected and to be reacted upon timely also during startup.

In detail, during congestion avoidance Vegas computes once every RTT the estimated number of packets Q in the queue at the bottleneck and reacts to this detailed congestion input according to an AIAD scheme: If Q is less than a threshold $\alpha = 2$, then cwnd is increased by one. If Q is greater than a threshold $\beta = 4$, then cwnd is decreased by one. Obviously there exists a steady state zone: If the queue estimation is in the range of two to four packets, no modification is applied.

For the startup phase, [120] proposes to use normal slow start only every other RTT to allow applying the delay estimation also in this phase. Actual implementations, notably the Linux kernel implementation, use normal, unmodified slow start.

Summarizing, the proposed Vegas algorithm results in a very low maximum queue size and a true steady system state.

2.4.3.5 uTP and LEDBAT

uTP [86] defines headers and behavior of the protocol that BitTorrent uses. uTP runs on top of UDP. LEDBAT [RFC6817] is the experimental standard created at the IETF that describes uTP's congestion avoidance behavior without specifying header formats. So with regard to behavior, uTP and LEDBAT are interchangeable protocols. The behavior of LEDBAT has been extensively examined, mostly by D. Rossi at TELECOM ParisTech [131, 108, 132, 133, 134], yet there is more research on LEDBAT also by other groups [135, 136, 137].

For the startup phase, uTP uses the same algorithm to calculate the cwnd as in the congestion avoidance phase. The LEDBAT Request For Comment (RFC) leaves open if slow start should be used or if the cwnd should be grown in a more moderate way. When used for background transmissions, slow start should not be used as argued before. The core congestion avoidance algorithm is based on OWD measurements, which eliminates the uncertainty regarding the crucial downstream delay that RTT-based algorithms such as TCP Vegas are facing. As TCP Vegas, uTP also defines a target queue size that in contrast to TCP Vegas is not measured in packets but in time, i.e. milliseconds. Moreover, this target queue size is configurable and does not need to be aligned between communicating hosts. The cwnd is then computed for each ACK received as also shown in Equation 2.5: The new cwnd is calculated by adding the relative difference between the target queue size τ and the measured queuing delay multiplied by a gain factor γ, divided by the cwnd.

$$cwnd = cnwd + \gamma * \frac{\tau - (OWD_{last} - OWD_{min})}{\tau * cwnd} \tag{2.5}$$

The queuing delay measurement is computed as the minimum measured OWD substracted from the last measured OWD and γ is required to be between zero and one. Moreover, cwnd is never allowed to grow more than a fixed threshold of ALLOWED_INCREASE during one RTT which is usually set to one or two packets. So, this algorithm increases the transmission rate until the target delay τ is met, and decreases if the measured delay is higher than τ. While this target delay is met, the cwnd is not changed.

If uTP / LEDBAT detects a packet loss, the algorithm acts like standard TCP, i.e. it halves its cwnd. This immediately implies that if the bottleneck's buffer cannot hold packets for at least the target delay τ, the protocol behaves just like NewReno.

Nevertheless, the available uTP implementation in libutp [138] as well as the RFC use 100 ms as default target delay (older drafts of LEDBAT proposed 25 ms, but the final RFC proposes "100 ms or less"). This means that such traffic induces a standing queue equivalent to 100 ms, which does not seem desirable since it results in operating like TCP NewReno for all buffers equivalent to less than 100 ms. Other research publications also suggest that it is necessary to use much lower targets for uTP or LEDBAT to properly work as background CCA and propose [137, 136, 139, 140] using a much lower target value. [137, 136, 139] propose to use 5 ms, but none of them evaluated LEDBAT's behavior for even lower values.

uTP allows to reduce the cwnd to zero. In that case, the sender ceases sending packets for one second, and then sends only one small packet (150 byte). We use the term "hibernation" to describe this freezing behavior. On the one hand, this behavior allows uTP to pose almost no load

on the network. On the other hand, the long delay of one second leaves bandwidth unnecessarily unused in many situations. Nevertheless, this is a proper design choice for a background CCA.

Summarizing, the uTP / LEDBAT CCA results in a very low maximum queue size and a true steady system state as TCP Vegas, uses more robust delay measurements and additionally implements a careful startup behavior and the hibernation behavior.

2.4.3.6 Compound TCP

Compound TCP [141] (now also being standardized in the IETF [142]), is a CCA developed and used by Microsoft. It aims at providing the synergy of delay-controlled and loss-controlled approaches. It uses slow start and halves its cwnd on packet loss as NewReno. To achieve synergy of delay- and loss-controlled CC, Compound uses a cwnd composed of the normal NewReno cwnd (here indicated by $cwnd_{NewReno}$ and a so-called delay window $dwnd$, see Equation 2.6.

$$cwnd_{Compound}(t) = cwnd_{NewReno}(t) + dwnd(t) \tag{2.6}$$

$dwnd$ grows rapidly if the queuing delay is zero, i.e. the link is underutilized. If the link is fully utilized and a queue builds up, $dwnd$ is reduced to zero, so then Compound behaves like NewReno. The delay-controlled fast increase makes Compound tolerating non-congestion packet loss much better than NewReno which is an important property for so-called "long fat pipes", i.e. connections with large BDPs. Specifically, Compound computes an estimation of the queue size $Q_{est}(t)$ and depending on its value defines $dwnd$ as shown in Equation 2.7.

$$dwnd(t+RTT) = \begin{cases} dwnd(t) + \left(\alpha * cwnd_{Compound}(t)^k - 1\right) & \text{if } Q_{est}(t) < \lambda \\ max\left(0, dwnd(t) - \zeta * Q_{est}(t)\right) & \text{if } Q_{est}(t) >= \lambda \\ \left(cwnd_{Compound}(t) * (1-\beta) - \frac{cwnd_{NewReno}}{2}\right) & \text{if loss is detected} \end{cases} \tag{2.7}$$

For the evaluation in [141] α is set to $1/8$ and k to $3/4$. The authors argue the need to use a threshold γ large enough to provide robust detection of the existence of a queue and choose a value of 30 packets. β is of course set to $1/2$ to achieve the same rate halving as NewReno. The authors do not publish the value of ζ they used for the evaluation but from the figures it should define a linear decrease to compensate the linear growth of $cwnd_{NewReno}$, resulting in an about constant cwnd for a queue size between γ and 2γ.

Summarizing, the resulting behavior achieves the goals of TCP Compound: TCP Compound flows provide good efficiency for large BDPs due to the aggressive increase while no queue has built up in congestion avoidance. This behavior also improves RTT fairness. The assumed linear growth beyond 2γ guarantees TCP fairness and avoids being pushed back by loss-controlled CCAs. The rather large threshold of 30 packets queue size indicates that Compound is rather designed for fast connections. For instance, at a 20 Mbit/s access link 30 full-sized 1500 byte packets represent 18 ms of queuing delay which is far more than needed for a robust detection if some queue exists.

2.4.4 Relation to Buffer Sizing

This section is about buffers able to hold several or many packets, which are managed by a queuing discipline or packet scheduler. Such buffers are a necessary function in any packet switched communication to compensate for short-term packet bursts, so they are needed at any interface that shall forward packets from more than one interface or from a faster interface. Moreover, buffers are also linked to transport layer congestion control. Due to the (necessary as has been proved [94]) massive reaction to congestion in AIMD CC schemes, such congestion-controlled traffic is vulnerable to cause underutilization of the bottleneck link after a decrease. Buffers help mitigating this issue by two aspects: First, if the buffer is rather large, the buffer helps to fill the bandwidth gap potentially left unused by the now reduced sending rate after a congestion event. The second effect if of higher importance: A rather big queue size at the time of a congestion event also means a correspondingly increased BDP and cwnd, so after halving the cwnd starts from a bigger value. The difference in impact of these two aspects is in the very nature of the AIMD principle: The first is just an additive compensation, while the second effect is a change in the multiplicative part of the rate adaptation. In consequence, without any buffers a single connection controlled by a pure AIMD CCA such as NewReno only achieves a 75 % bottleneck utilization because the typical sawtooth is fully effective on the bottleneck link. In contrast, if a buffer of one BDP is available at the bottleneck, the sawtooth pattern effects the buffer utilization only and does not affect the bottleneck link at all, resulting in 100 % utilization. Note that for CCAs with a greater β, such as TCP Cubic, the buffer required for full utilization is less than one BDP.

If a bottleneck is shared by N TCP flows, the buffer size required for full utilization depends on the synchronization of these flows. If there is perfect global synchronization, the frequency of the sawtooth pattern is increased but its height is not changed, so again one BDP of buffer is required. If all flows are perfectly minimally synchronized, the buffer required may be as low as $\frac{1}{\sqrt{N}}$ as Appenzeller et al. showed [143].

Unfortunately, ISPs usually neither know the number of flows a link carries, nor their average BDP and most AQMs do not achieve minimal synchronization. Therefore, networks are mostly configured based on pessimistic assumptions. Already in 1994 a buffer size of one BDP was recommended [144]. And still today, there is the rule of thumb to provide one BDP of buffer based on a rather pessimistic RTT estimation, e.g. [145] suggests calculating with an RTT of even 250 ms. In this context, it is important to consider that for each flow, i.e. also for each TCP connection, there is only one effective bottleneck at a time in static scenarios. So, a flow may pass several interfaces where multiplexing results in small queues being built up from time to time, but the growing queue provoked by the rate increase of TCP occurs at only one place in the network, the (first) link with the lowest capacity on the path. Due to the architecture of networks, see Section 2.1.1, this usually is the access link or during peak periods maybe some aggregation link in the regional access network. In both cases the decisive queue is the per-subscriber queue in the Hierarchical Fair Scheduler (HFS). Since bandwidth is still the major selling point for broadband access services, the ISPs usually take no risk of complaints and dimension the buffers based on rather large RTT estimates.

Recent measurements show that the average buffer size in the Internet is below 100 ms, at about 70–80 ms [146, 147]. Nevertheless, they also show that a significant number of buffers is well

larger than 100 ms. There even is the term “bufferbloat” used in the community for far too large buffers [148]. So, there are buffers sized more optimistically than the wide-spread rules of thumb, but nevertheless buffers are still dimensioned for delays of long distance connections, some rather for intercontinental connections than for regional connections.

Concluding, purely loss-controlled AIMD CCAs need buffer at the bottleneck to fully utilize that bottleneck. Today, ISPs configure rather large buffers allowing full utilization also for unfortunate conditions such as large RTTs. This also applies to the per-subscriber buffers of the packet schedulers at BNG downstream interfaces. Since bandwidth is the main selling point for broadband Internet access, the ISPs are not expected to significantly lower the configured buffers of the average subscriber in the near future.

3 Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload

This chapter is dedicated to Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload (RADICCO), the algorithm developed and evaluated in this thesis. First, it explains the motivation of this thesis and outlines the technical problem statement. Based on this, qualitative design objectives and quantitative performance objectives are derived. With this background, we present the high-level concept of RADICCO. In continuation, Chapter 3 presents related work that aims for similar goals. As a foundation of RADICCO, we detail the assumptions and prerequisites RADICCO is based on and discuss their future validity. Then, we present the algorithm of RADICCO and introduce the calculations for internal states and the effectively allocated rates. Finally, core design decisions and potential alternatives are discussed.

3.1 Motivation

The core motivation for RADICCO is founded on five facts about today's regional access networks:

- Aggregation links in hierarchical access networks constitute bottlenecks at times of peak and overload. In many access networks, such overload occurs daily.
- The load of access networks varies heavily during a day and peak load applies to just a few hours at most.
- At any time, the downstream traffic in access networks contains a significant amount of deferrable traffic, which does not contribute to immediate QoE and is called background traffic.
- A large share of this background traffic uses special CCAs and thus behaves differently than foreground traffic.
- The bottleneck for most of the downstream traffic is located in the access network, i.e. in its BNG's packet scheduler.

The challenge is to develop an efficient resource allocation system exploiting these facts to reduce resource allocation of background traffic during times of overload in order to favor the foreground

traffic. First, this requires recognizing the background traffic as such. Traffic differentiation by CC results in different behaviors at the bottleneck. Therefore, traffic differentiation by CC can be detected and considered at the BNG's packet scheduler. Second, resource allocation at the BNG downstream interfaces is carried out by the packet scheduler, so this is component to enforce the adaptation in resource allocation.

The requirement that traffic behavior shall be used for traffic differentiation is crucial. First, this signaling origins from the subscribers' communication partners, i.e. any explicit signaling could not be trusted. Second, there are strong incentives to use a background CCA for background traffic: Since most of the time the only competing traffic is the subscriber's own traffic since the bottleneck is the access link, not using a yielding background CCA could result in substantial drawbacks (see Section 3.6).

The effect aimed for by the developed resource allocation system compared to a non-adapting, neutral hierarchical scheduler, i.e. a HFS, can be described from three perspectives:

- From the ISP's perspective, the traffic composition at times of overload is shifted towards foreground traffic. Consequently, the traffic composition after times of overload is shifted towards background traffic.
- From a foreground receiver's perspective, there is more capacity available during overload which on average allows to achieve a better user experience.
- From a background receiver's perspective, there is less capacity available during overload which on average delays transmissions but at maximum by the duration of overload. Due to the nature of background traffic, this does not result in a disadvantage in terms of QoE.

So, compared to a HFS, a scheduler adapting resource allocation in the described way results in advantages for both the ISPs as well as the users, i.e. the subscribers, which are listed in the following.

Resulting benefits for ISPs:

- Smoother utilization of infrastructure.
- Higher average utilization of infrastructure.
- Deferred and reduced pressure for investments in infrastructure.

Resulting benefits for subscribers:

- Increased QoE during peak and overload periods.
- Diminished impact of events of exceptionally high resource demand on QoE.
- A (slightly) increased traffic volume since more interactive traffic such as VoD is delivered with higher quality and, consequently, traffic volume.

For subscribers, the first benefit is the by far most important.

A system for adapting resource allocation based on distinguishing foreground and background traffic achieves these effects for the usual daily peak load as well as for exceptional peak loads such as caused by events of high public interest, e.g. a sports event but also a natural disaster.

3.2 Problem Statement

Any proposal for a system implementing such adaption of resource allocation based on distinguishing foreground and background traffic must provide:

1. A mechanism for recognizing the traffic type.
2. A means to adjust the resource allocation.

The mechanism for recognizing the traffic type should recognize the traffic type reliably and fast. In particular, it must react fast to traffic changing from background behavior to foreground behavior.

The mechanism for adaptation of resource allocation must work on all aggregation levels, this means its must be able to adapt allocation among the access links fed by an AN and among the aggregation links fed by an AGS. The adaption must consider the different ways foreground and background CCAs work in its actions. Moreover, it should minimize interference with the traffic type recognition algorithm. Since resource allocation at the BNG downstream interfaces is carried out by a hierarchical packet scheduler, this mechanism either defines a new hierarchical packet scheduler or extends existing ones.

The combined system must allow implementation, i.e. it must either allow implementation in a single device or define appropriate signaling to allow distributed implementation. In particular, for an integrated implementation it is required to consume sufficiently little computational resources.

3.3 Concept of Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload

The core concept of RADICCO combines two functionalities: First, it recognizes foreground and background traffic on subscriber level. Second, it adapts the scheduling weights of the hierarchical scheduler during peak load so that the foreground traffic receives more bandwidth and the background traffic less.

For recognition of foreground and background traffic, RADICCO does not introduce any explicit signaling but relies on recognizing the different behaviors of foreground and background CCAs. RADICCO operates on subscriber, i.e. access link level. This is possible since a foreground CCA is known to dominate an aggregates behavior at the bottleneck (see Section 2.4.2). Therefore, we also use the terms foreground subscriber and background subscriber for a subscriber receiving traffic of the respective type.

Relying on the senders' dynamic behavior forces RADICCO to dynamically operate in both traffic type recognition and weight adaptation: The behavior of subscriber's traffic may change at any instant and such change should be recognized and honored by RADICCO as fast as possible. Rate adaptation must take into account the senders' control loops as well as RADICCO traffic type recognition. To avoid wasting bandwidth and oscillations, the rate adaptation must be carried out smoothly.

By controlling the scheduler's weights depending on the traffic behavior, RADICCO adds another control loop to the overall system of rate control: The senders execute a control loop on their sending behavior according to their CCAs and RADICCO at the hierarchical scheduler executes a control loop on the subscribers' weights, i.e. their allocated rates. RADICCO's control loop does not operate on transport layer connections, as do CCAs implemented by TCP and uTP, but operates on per-subscriber traffic aggregates.

The interaction of these control loops and the involved feedback delay result in a conflict of objectives for RADICCO:
On the one hand, the traffic type detection benefits from more static bandwidth allocations. The input for RADICCO's traffic type recognition is limited to observing capacity and buffer utilization. This allows distinguishing foreground traffic from background traffic well for a static system, especially for a static bottleneck capacity. If the bottleneck capacity is reduced, the recognition becomes less reliable. The reason is that, since RADICCO does not know the RTT, i.e. the delay in the CCA's control loop, every reduction of the bottleneck capacity is followed by a period in which it is not clear how to interpret an increasing queue size: Either the sender still uses a background CCA but could not yet react due to the RTT delay in its control loop or a foreground CCA took over control and ignores the increasing delay. The faster the reduction is, the greater this uncertainty in traffic type recognition becomes, so the less reliable it can be. So, from traffic type recognition perspective, a connection's bottleneck capacity should only be changed slowly and carefully to provide good traffic type recognition.
On the other hand, the benefit of RADICCO depends on reducing capacities allocated to background subscribers. So, the larger this reduction of capacity and the faster it is applied, the more capacity is freed for foreground subscribers. Moreover, background transfers may have arbitrary sizes and the traffic type recognition algorithm may result in temporary false recognition of background subscribers. Therefore, it is even more important to rapidly reach a low capacity allocation for a background subscriber after its recognition as such. So, from this perspective, a background subscriber's capacity should be reduced fast to achieve shifting more bandwidth from background to foreground traffic.

RADICCO reduces the weight of a background subscriber by a constant amount per served packet and maintains a minimum weight. By the per-packet weight reduction, RADICCO only carries out weight reduction if it is possible for the sender(s) to become aware of it, even if indirectly via the receiver. The minimum weight prevents starvation and should protect services of low data rate such as a VoIP call from being hit by increased delays.

Another principle in RADICCO is to fill up rates: If reducing the weights of background traffic resulted in underutilization of the shared aggregation interface, RADICCO distributes this headroom among the background subscribers in a proportionally fair manner.

3.4 Objectives of Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload

In order to be deployed, or attract further research in the first place, RADICCO should meet absolute requirements, i.e. qualitative objectives, as well as perform well in terms of specific

quantitative objectives. We will provide an overview of relevant objectives for both categories, followed by short definitions and reasoning for each objective.

3.4.1 Qualitative Objectives

The qualitative requirements are:

- Network neutrality
- Sufficient efficiency
- Smooth Rate Allocations for Foreground Traffic

3.4.1.1 Network Neutrality

RADICCO is based on distinguishing background and foreground traffic and treating them differently, what raises the question of network neutrality. Network neutrality describes the principle that the network treats all traffic equally. Depending on context, this principle is interpreted differently. Largely, the term implies that the network shall not decide on which traffic shall receive better or worse service. Many countries regulate their ISP market dictating network neutrality in that interpretation. For instance, the FCC demands ISPs to perform no blocking and no throttling based on "legal content, applications, services, or non-harmful devices" [149]. Moreover, they demand that any mechanism in today's Internet should be non-discriminating, i.e. there should be no type of service generally treated better than another on account of the ISP's decision. We use the term in this sense.

We think that a solution to the problem described in Section 3.2 does not have to and should not depend on the network, i.e. the ISP, deciding on priorities. Therefore, we claim that proposed solutions should not violate network neutrality based on this definition.

3.4.1.2 Sufficient Efficiency

The efficiency of any mechanism developed for a certain purpose must allow to be implemented in real equipment and deployed for the intended purpose in real networks. Since RADICCO is an algorithmic extension to a scheduler, its computational complexity adds to the one of the scheduler and the total complexity must still allow implementation. We assume the efficiency of RADICCO being sufficient if it has the same or lower asymptotic computational complexity than potential schedulers to be extended.

3.4.1.3 Smooth Rate Allocations for Foreground Traffic

As argued in 2.3.3, end-to-end rate control is prevalent in today's Internet. The underlying control loops work better and more efficiently the more static the properties of the connecting network are, especially the bandwidth available and the RTT. RADICCO does introduce new

dynamics in the system in that it dynamically changes the bandwidth of subscribers at peak load, primarily of background subscribers and in consequence also of foreground subscribers. Nevertheless, these induced changes may be steeper or smoother and may have local impact only or concern the whole system. In particular, a change may affect small parts of the system, e.g. only single subscribers, or affect the BNG interface as a whole. Moreover, the end-to-end control loops of the senders' CCs may amplify the effects of a change event. This may even result in oscillations that should be avoided.

Steep changes, i.e. changes of substantial extent within a short period, cannot be handled well by the sender's CCA. Therefore, any such change may result in either an excessive rate reduction or in the queue running empty and thus underutilization of the allocated capacity. A rapidly changing throughput on transport layer may even trigger undesired reactions of higher layer control loops, e.g. of the control function of a DASH VoD streaming session. So, volatility in the allocated capacity of foreground subscribers comes with a risk of QoE degradation. For background subscribers, this does not apply since background traffic is assumed to not contribute in short-term QoE (see Section 3.6.3).

We consider this a qualitative objective since this objective can be considered in the design, but is hard to measure quantitatively. Moreover, the changes occurring during execution heavily depend on the used CCAs whose control loops unavoidably interact with the control loop of RADICCO.

Therefore, RADICCO shall be designed to avoid steep changes in the allocated rates, primarily for rates of foreground subscribers.

3.4.2 Quantitative Objectives

Due to the very nature of RADICCO as a system implementing service differentiation, the quantifiable performance goals have a strict order of priorities. In the following they are listed in order of decreasing importance.

1. QoE improvement.
2. High bottleneck utilization, i.e. high bandwidth for background traffic.
3. Fairness among foreground subscribers.
4. Fairness among background subscribers.

We discuss each goal in the following.

3.4.2.1 Improved QoE

The paramount goal of RADICCO is to increase QoE. QoE is a user-centric metric and therefore cannot be measured universally. Nevertheless, the impact an ISP has on a user's QoE directly depends on the QoS the ISP delivers and the QoS requirements of the respective service.

Regarding QoS, increased bandwidth and reduced delay are generally considered having non-negative impact on QoE. Nevertheless, a QoE improvement caused by increased bandwidth or reduced delay can only be assessed specifically for a service. The better a service and its requirements are known or understood, the better a receiver's QoE can be estimated. But often even the service provider does not know the exact utility function of his services, i.e. the function that maps provided QoS to QoE. Nevertheless, such a function theoretically exists: For instance, for a VoD service that uses DASH (or similar technology), it maps the available bandwidth to a resolution and quality, which in turn map to a QoE level.

Nevertheless, also without utility functions there are distinct types of services for which we know the general relationship between QoE and single QoS parameters as well as the most important QoS requirements. In this thesis, we estimate RADICCO's impact on QoE for four service types represented by statistical traffic models.

Note that background traffic by definition does not contribute to short-term QoE since the sender can switch to a foreground CCA at any time. So, background traffic is neglected in terms of this objective.

3.4.2.2 High bottleneck utilization

The basic motivation to develop and to deploy RADICCO is temporary overload on aggregation links in the regional access network on a daily basis, i.e. the temporary existence of bottlenecks at these links. These bottlenecks limit the possible data rate for the active subscribers to less than their nominal rate. If RADICCO results in lowered utilization, it extends these times of overload. This may be acceptable to some extent if the achieved QoE is improved, but generally a maximum utilization should be aimed for.

Since RADICCO approach to increase the QoE of foreground traffic is increasing the bandwidth assigned to foreground traffic, a high bottleneck utilization also means filling the remaining bottleneck capacity with background traffic.

3.4.2.3 Fairness among Foreground Subscribers

Today's HFSs enforce a relative guarantee among all active subscribers, namely that any one of them receives a share proportional to his weight. While we aim at allocating more than this fair share to a foreground subscriber, we aim to not relinquish this relative guarantee within the group of foreground subscribers.

Since RADICCO does not alter the weights of recognized foreground subscribers, this goal effectively requires in checking, first, the background traffic recognition for bias and, second, the whole system for potential synchronization between RADICCO control loop and the background CCA control loop.
If the background recognition algorithm treats all subscribers equally at all times, all subscribers receiving similar traffic, e.g. per the same traffic model, will suffer from being falsely recognized as background traffic to about the same extent.

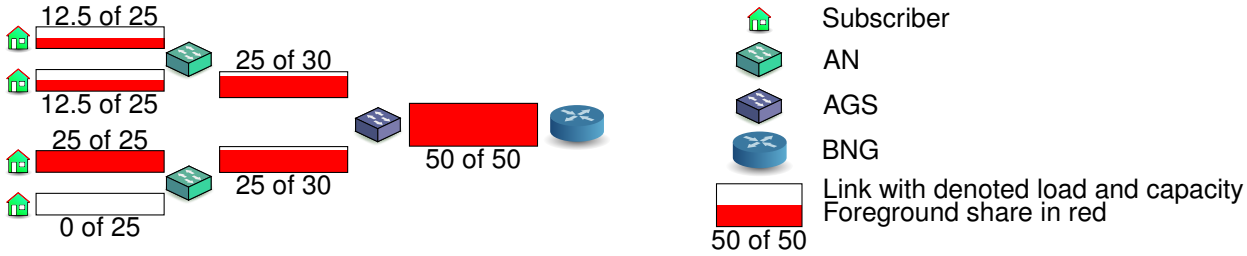


Figure 3.1: Illustration of a locally fair foreground allocation that is not globally fair

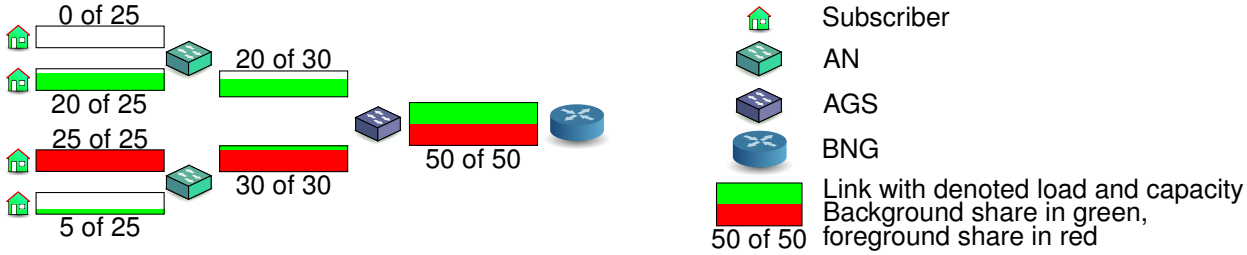


Figure 3.2: Illustration of a desirable allocation with highly unfair rate allocation among background subscribers

We aim for the same fairness as achieved by a HFS, i.e. a local fairness at each bottleneck only. In contrast, the following situation does not violate achieving this goal: Consider a three-level topology with the BNG interface as well as one AGS interface being overloaded. In that case, a foreground subscriber served by a non-overloaded AGS interface receives higher bandwidth than a comparable foreground subscriber served by the overloaded AGS interface. See Figure 3.1 for an illustration of a locally fair rate allocation in such scenario. Enforcing fairness among all leaf nodes of one type in such situations would violate the first and the second objective. Clearly, in such situations we accept limiting fairness among subscribers in favor of higher foreground rates and higher bottleneck utilization.

3.4.2.4 Fairness among Background Subscribers

We would also welcome fairness among background subscribers, although this goal is of lowest priority for several reasons. First, background traffic is not contributing to short-term QoE, so the allocated bandwidth does not matter to the user. Second, background CCAs are much less predictable than foreground CCAs. Foreground CCAs always keep the queue non-empty if we do not increase the effective rate very fast. Thus, foreground subscribers are permanently active from the scheduler's perspective. This cannot be expected for background subscribers: Especially, the low increase speed of all background CCAs in congestion avoidance and uTP's hibernation behavior may cause a subscriber's queue run empty frequently. Third, in topologies with at least three hierarchy levels this secondary objective often contradicts the primary objective of increasing foreground QoE by increasing rates of foreground subscribers. In these networks, in a subtree with a high fraction of foreground subscribers the bandwidth allocated to a background subscriber must be lower than in a subtree with a lower fraction of foreground subscribers. Figure 3.2 shows an illustration of a scenario, in which the priority for foreground traffic increases unfairness among background subscribers. Both the first and second quantitative objectives are perfectly met by the depicted allocation. Note that we again aim for local fairness

at every bottleneck only.
Summarizing, this goal is of least importance but it would be nice to achieve it.

3.5 Related Work on Peak- and Overload Management

In this section, first, fundamental terms are defined and the context of overload management systems is discussed. In the following, we shortly present two approaches to that challenge that are widely used by today's ISPs and two approaches recently discussed and published in the research community.

3.5.1 Definitions

RADICCO aims at allocating less bandwidth to background traffic when there is not sufficient bandwidth to supply every subscriber with its access link bandwidth. In this context, the terms *congestion*, *overload* and *peak load* are often used, but the definitions used in the network community unfortunately vary, often overlap and sometimes even contradict each other.

In the following we will give short definitions for these three terms. Note that although these terms mostly refer to the state of a transmit interface, they are often transferred to the respective links, e.g. "overloaded link". Obviously, a link can just be loaded with as many bits as it can transmit, but for point-to-point links as used in hierarchical access networks each directed link corresponds to a one-to-one counterpart transmit interface, so there is no room for ambiguity.

Definition of Congested

The term *congestion* is linked to end-to-end rate control, typically CCAs, so we use this term to emphasize the feedback given to an end host. For a definition, see Section 2.3.1.

Definition of Overload

For interfaces serving several subscribers, such as BNGs' and aggregation switches' downstream interfaces, we use the term *overload*, emphasizing the (undesired) network state rather than its meaning for the end hosts. Overload is defined as a state where there is more load than the interface can handle, i.e. more packets available for transmission than the interface can transmit. This means that a queue builds up, either directly at the interface or in the (potentially remote) scheduler.
Although the overload definition differs only in a detail from the congestion definition, its role and its meaning is fundamentally different: When an access link is congested, this typically means that CC is at work, probing for bandwidth limits. So, congestion is a normal and somehow desired state detected by an end host. In contrast, when an aggregation link is overloaded, subscribers do not receive their full contracted capacity, but their bandwidth is limited by a concealed bottleneck within the ISP's network. So, overload is an undesired, hopefully unusual

state. Usually, only the ISP can safely detect overload in its access networks and distinguish it from overload at a peering point or at the sender. So the term overload is linked to an network operator point of view. Overload is a state difficult to detect since due to the dynamics in offered load and the elastic nature of today's Internet traffic, queues at aggregation links often run empty for short periods even at very high loads. So, often less strict state definition of critical load is desired. The term peak load is such a term.

Definition of Peak Load

Peak load means critical, undesired high load at an interface serving many subscribers, i.e. it is a term from ISP perspective, too. There is no universal definition of peak load, but all technical definitions base on medium-term averages. These average load values may be based on strict, disjunct intervals, as for Comcast's approach (see Section 3.5.3), or calculated as EWMAs. By definition, peak load includes overload and usually also covers the periods before and after overload phases. Therefore, peak load detection is often used to trigger counter-measures against overload. This allows acting pro-actively and early, often preventing longer overload phases. For RADICCO, we only use peak load to trigger different modes of operation. In our implementation we use the EWMA of the interface's load and two predefined thresholds for peak load detection with hysteresis: If the EWMA of the interface's load is above the high threshold, we consider this interface being under peak load, if it falls below the low one, we consider it not under peak load.

3.5.2 Context of Overload Management

Managing peak- and overload has been approached for various motivations and thus in fundamentally different ways.

All four systems presented in the following have been designed to solve a challenge that many ISPs are facing: A small fraction of subscribers, sometimes called heavy users [102], continuously place much higher demands on network resources than most subscribers. This becomes an issue to ISPs during peak periods, when resources are driven to their limits. When the first proposals came up, these heavy users mostly used P2P services for file sharing, often receiving and transmitting data all day. Standard HFS are perfectly fair at every instant, but do not take into account past allocations. Therefore, such heavy users receive their instantaneous fair shares at any time, also during times of peak load. Because of this, other subscribers, although using their Internet access only sporadically, may not receive maximum capacity, i.e. the capacity they pay for, during times of peak load. This situation is not deemed fair and results in dissatisfied (light user) customers. So, all the presented approaches aim for a fairness different from the instant fairness a fair scheduler aims for. These approaches take into account past behavior of a subscriber and thus may reach a better medium-term fairness than a HFS.

A general issue is the relation between managing overload well, i.e. minimizing its impact on the users' QoE, and upgrading network elements, i.e. adding capacity. Repeatedly, comments are read that managing overload should not be examined at all but the ISPs should just upgrade

their network infrastructure appropriately. Yet, there are good reasons for overload to occur in communication networks:

1. Transported traffic is ever increasing and at some point eventually some peak loads are beyond the capacity of the respective infrastructure.
2. There are unpredictable events that cause a drastic increase in load, for instance a natural disaster.
3. The service offer of any ISP must be affordable for the customers and competitive.

So, as long as there is no significant impairment for the users (or there are no other ISPs offering better service), there are economic reasons not to invest into infrastructure. Finally, fourth: It is possible to manage load so that no user is negatively affected in terms of QoE. This is possible because overload usually concerns only a short time of the day, so there is a lot of headroom during other times of the day. Then there is traffic, e.g. P2P file sharing and software update downloads that may be deferred or regulated down without perceptible consequences.
So, the capability for managing overload is necessary even in not undersized networks and there are good reasons not to add capacity as long as overload is only temporary and can be effectively managed.

3.5.3 Comcast's Protocol-Agnostic Congestion Management System

Description

Comcast's congestion management system [RFC6057] is a network resource management system aiming at implementing a resource allocation in peak periods that is perceived as a fair allocation. In contrast to our definition, in [RFC6057] the term congestion is used to describe a medium-term network state that corresponds rather to our definition of peak load than to that of congestion (see Section 3.7.1).
Comcast's Protocol-Agnostic Congestion Management System is based on, first, continuous monitoring of capacity usage on subscriber and aggregation links in the network and, second, adapting priorities of some subscribers' traffic. Therefore, all functions must be located at or near to the BNG (called regional network router in [RFC6057]). All monitoring is carried out based on fifteen minute intervals, which is huge compared to changes in congestion. If the measured load reaches a threshold level (70 % in upstream, 80 % in downstream) for some aggregate link, all subscribers served by that link, i.e. whose traffic contributes to that load, are examined more closely. If a subscriber has used more than 70 % of his contracted capacity in the respective transmission direction during the last measurement interval, its traffic is assigned to a lower priority. Comcast uses the terms *Priority Best Effort* and *Best Effort*, but these classes effectively correspond to the BE traffic class and a class of even lower priority. A subscriber assigned to the lower priority class is re-assigned to normal BE as soon as the subscriber's consumption has declined below a lower threshold of 50 % of their contracted bandwidth during a whole measurement interval. This system uses strict priority between the two classes: Packets of the lower priority class are only transported if there is no packet of higher priority available.

Discussion

Comcast's approach is named Congestion Management, but is rather a peak load management, which is purely based on rate measurements and does not take into account any congestion by any meaning used by the transport layer community. Nevertheless, it aims at providing a solution to a problem similar to the problem identified by us. There are crucial differences regarding the problem definition and the Comcast approach has fundamental shortcomings. For one, this approach does not take into account any priorities possibly implicitly signaled by the end hosts. It basically assumes that large transmissions are less relevant for user experience than lower bandwidth transmissions. Second, this system reacts very slowly, its loop works on fifteen minute intervals. Moreover, it is based on averages only. While this might have been appropriate for the P2P scenario mainly targeted at by the authors of [RFC6057], this is not sufficient for many other scenarios. For example, a node receiving and installing software updates often produces load that is below the mentioned thresholds, but is receiving bursts of heavy load that could be allocated less bandwidth without deteriorating QoE. Third, this system takes no measures to prevent starving single subscribers, i.e. it might allocate not bandwidth at all to some subscribers for significant periods. We consider this being generally not desirable.

Generally, this approach is slower and more coarse-grained in time than desirable, resulting in potentially big false positive intervals as well as false negative intervals. On the one hand, if a subscriber made use of its access link above the thresholds, all its traffic in the next interval will receive low priority regardless of volume or type. On the other hand, a subscriber may use heavy bursts of traffic for the bigger part of the time without being classified to low priority if he on average consumes less bandwidth than the threshold.

So, while the proposed rate management may work well for some scenarios such as the targeted P2P users, it neglects other potentials for reducing bandwidth of low priority traffic. It also ignores useful information provided by senders such as the implicit priority signaling by using different CCs.

3.5.4 Traffic Management based on Deep Packet Inspection

In this section, we do not discuss a single and well documented traffic management system, but we will sum up core properties of systems that are widely used but usually poorly documented.

Description

Traffic management systems based on DPI, e.g. Sandvine's "Quality Guard" [150], are all based on classifying every packet by inspecting packets. Every incoming packet is parsed to identify the transport layer connection it belongs to. Usually a deep inspection also of application layer content is carried out for every new connection detected, i.e. the first packets of any newly established connection. Based on the findings of this deep inspection the connection is assigned to a service type. Due to this fundamental step, we summarize all systems using this technique as traffic management systems based on DPI. For each service type the deploying ISP configures a priority. For instance, Sandvine recommends to configure service priorities based on tolerance

for latency. The derived service priority of a flow is then used for enforcing traffic management policies, which may take many forms and may be enforced at a separate node. Usually, policy enforcement is only applied if the network's load is deemed critically high. One example for traffic management policies is to use service priorities to assign a scheduling priority lower than the BE priority to a set of services, the set being adapted to the current level of overload.

The core step in such systems is the packet inspection. It requires keeping state per transport layer connection. The inspection is a so-called DPI: That means parsing some first packets (the actual number often varies from service to service) of every new connection down to the details of the application layer, e.g. not only IP, TCP and Hyper Text Transport Protocol (HTTP) may be parsed and evaluated, but also the type or names of the transported objects.

Discussion

Compared to Comcast's approach, DPI-based traffic management is much more fine-grained in several dimensions: It is not based on fixed (long) intervals but changes state when a connection is set up. Moreover, working on the level of transport layer connections rather than subscriber level allows some traffic receiving normal priority while another connection of the same subscriber, be it to the same end device or not, receives reduced priority.

The DPI-based classification is very expensive in terms of implementation complexity, computational effort, and consequently also in terms of Capital Expenditure (CAPEX). Nevertheless, no DPI is flawless, these mechanisms have non-zero false-positives and false-negatives. False recognitions of foreground traffic may have severe impact on QoE since the inspection is carried out only once per connection. Moreover, such system cannot prevent being evaded: services widely being assigned low-priority such as traffic of P2P applications can switch the used TCP ports and use encryption to effectively prevent reliable identification.

This approach does also not seek cooperation with end hosts and does not consider any explicit or implicit signaling from senders or receivers. DPI-based traffic management systems depend on the subscribers communicating rather openly (at least sufficiently open to allow guessing used services) but do not implement any incentives to do so. In contrast, the behavior of such systems is completely defined by the ISP's configuration of service priorities and policies. This configuration and even the fact of deployment of such technology is usually not made public, so public debate is muffled. Nevertheless, pushing back traffic selected by such an operator-defined list of service priorities fundamentally violates the network neutrality principle. Moreover, such system may always be evaded, which invites starting a cat-and-mouse game between developers of services assigned low priority by the ISPs and the developers of the DPI traffic management solutions.

Summarizing, DPI-based Traffic Management is a more fine-grained and much more expensive tool to manage traffic than Comcast's Congestion Management System. It depends on subscribers behaving cooperatively, e.g. not using encryption and using standard ports, but neither gives them a say in priorities nor does it honor the subscribers' priority signaling in terms of choice of CCA. Moreover, it is generally prone to recognition errors and its core idea fundamentally violates network neutrality.

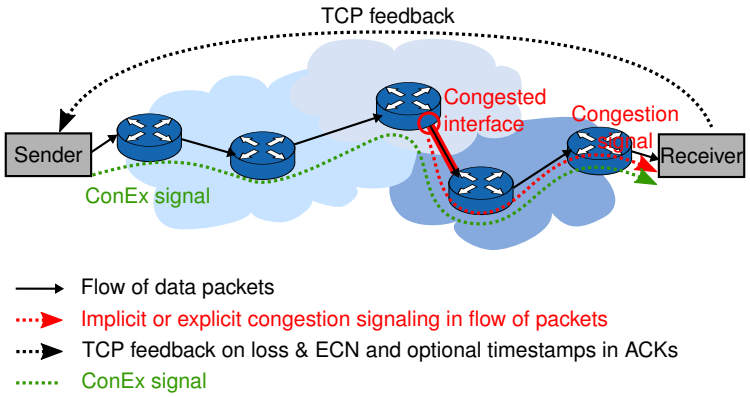


Figure 3.3: Signaling of a ConEx-enabled TCP connection relevant for the ConEx system

3.5.5 Congestion Policing based on Congestion Exposure

Another proposal for congestion management is Congestion Policing based on ConEx (CPC). This concept has been proposed by Bob Briscoe [151] and several researchers, including the author, have implemented and evaluated policing architectures based on congestion information made available by Congestion Exposure (ConEx). Unfortunately, these evaluations did not show the results hoped for [152], and in several cases have not even been published as we learned through private communication. Nevertheless, we shortly present the concept since the ideas behind that concept are worth noting and inspired the development of CPQs (see Section 3.5.6).

Description

Basically, the proposed system is based on per-subscriber policers that monitor the congestion a subscriber causes over time and penalize subscribers when causing excessive congestion. In the context of this system, congestion is understood as packet loss and ECN marks only. These signs of congestion are only detectable downstream of the bottleneck, i.e. only the receiver of a flow knows for sure the whole extent of congestion along the path of that flow. This information is meant to be published to every other node on the path, among them the policing entity, by a mechanism called Congestion Exposure (ConEx). ConEx is a mechanism allowing a receiver in a bidirectional communication to insert information on the downstream congestion to the upstream packets, and the sender to re-insert information on recent congestion into new downstream packets. The signaling important in a ConEx-enabled TCP connection is shown in Figure 3.3 . This congestion information is more detailed than TCP's duplicate ACKs that result in a full cwnd of duplicate ACKs regardless of the number of packets lost in the last RTT.

In consequence, all nodes on the downstream path, which may differ from the upstream path, learn about recent congestion on the whole path. Since they already see congestion upstream of their position, they now can deduct the amount of congestion downstream of their position on that path. ConEx was discussed and designed in a now concluded designated IETF working group [153], which produced some informational RFCs on concepts and use cases [RFC6789, RFC7713, RFC7778] and two experimental RFCs. The latter describe a modification to TCP [RFC7786] and an extension to IPv6 [RFC7837], allowing a fully working system for TCP traffic.

Discussion

When examining CPC it is important to keep in mind that congestion, i.e. losses, depend on the available bandwidth at the bottleneck for most TCP CCAs. The lower the available capacity, the shorter congestion epochs become, so the higher the absolute loss rate becomes. Moreover, for relevant BDPs the number of losses per congestion epoch, the loss rate, is very low: Consider a link of 24 Mbit/s capacity, 100 ms RTT and 100 ms worth of buffer at the bottleneck. For 1500 Byte packets, this means that the base BDP is 300 kB or 200 packets, and at full buffer the BDP is 600 kB and 400 packets. If one TCP NewReno connection utilizes this link, it needs 200 RTTs to increase the cwnd from base BDP to maximum BDP, transmitting 201 BDPs of packets during that time. This results in one packet loss every 40200 packets, or, considering the linear growth of the RTT, every $200 \cdot 150$ ms, i.e. every half minute. For a 2.4 Mbit/s link and same other parameters, we get one loss every three seconds. Another example highlights the weakness of ConEx with today's congestion semantic: At startup, due to the exponential slow start phase, the flow on the 24 Mbit/s link will cause about 400 packets to be dropped, more than would be dropped in six hours of continuous transmission. So, measuring a subscriber's contribution to overall load by measuring congestion is impossible with the current congestion semantics. This is expected to be different for more fine-grained congestion signaling with higher signaling rates. An inspiring example is DCTCP [117] that uses ECN signaling with adapted semantic. We examined this protocol and found that with some changes to Internet routers it could also be used in the Internet [154]. If such approach would become the dominant CC in the Internet, CPC might become an interesting overload management mechanism.

Another challenge with CPC is penalizing subscribers that shall reduce their rate. A pure policing element, i.e. which does not modify the scheduling, may only penalize a subscriber by delaying or dropping packets, which equals emulating congestion. Here again the congestion semantic poses a problem since any packet drop usually results in a drastic decrease in the sending rate, so the policer is likely to cause an undesired over-reaction.

Aside from open challenges in system design, there are also fundamental issues regarding the concept of CPC. First, ConEx does not work for unidirectional transmissions. This is a fundamental shortcoming which in reality is not of big importance since there is close to no true unidirectional traffic in today's Internet (see Section 2.3.3), in particular in the targeted residential access networks. Second, ConEx is a global signaling. It fully works only if all senders and receivers implement the protocols. This cannot be expected to happen in the near future. Nevertheless, CPC would work quite well and could be beneficial if receivers and senders of most of the traffic implement the protocol. But even that must be doubted.

On the positive side, CPC honors priority signaling of end hosts by using foreground or background CCs, even if indirectly: Since background CCs use only little buffer and aim to avoid provoking packet drops at all, such transmissions do not cause congestion in the definition used by ConEx. So, they might be transported even when a subscriber is to be penalized, depending on the design of the penalty function.

CPC reveals specific weaknesses if deployed at the root of a hierarchical access network and evaluated against the objectives defined in Section 3.4. In that use case, it makes a crucial difference that ConEx does not monitor the place of congestion. With some effort, a ConEx policer can calculate (or estimate well), which amount of congestion was experienced within

the regional access network and which outside. This is possible since the congestion occurred within the access network equals to the congestion signaled by ConEx minus the congestion seen one RTT ago (applies to both upstream and downstream). Nevertheless, a ConEx policer cannot distinguish if the congestion two subscribers experienced occurred at a shared link or at their respective access links. So, it also accounts for irrelevant congestion, produces false positives and in consequence unnecessarily penalizes subscribers, possibly inducing an unnecessarily low utilization.

In the use case of access network overload management, CPC shows another fundamental shortcoming: CPC does not carry out resource allocation itself, so it must be deployed combined with a queuing discipline or a scheduler. While for meshed networks a simple shared queue with an AQM is often a good choice, for gateways to hierarchical networks such as the regional access networks ISPs usually cannot give up all the guarantees a per-subscriber scheduling provides. Nevertheless, when combined with a standard HFS, ConEx adds a medium-term fairness between foreground subscribers, but the HFS still allocates a full fair share to all background subscribers and prevents background users from detecting the overall congestion on shared links. Therefore, such a combination cannot reduce background traffic during overload or peak hours. So, CPC is not capable of compensating the drawbacks of HFS and achieve better performance measured against our goals.

Summarizing, monitoring the congestion a subscriber's traffic causes allows to distinguish aggressive heavy users from more cooperative ones. This may provide a good basis for defining priorities or penalties aiming for some kind of medium-term fairness. However, the CPC approach has fundamental drawbacks and, moreover, must be combined with a resource allocation scheme. For the use case in focus, this probably must be some HFS. Unfortunately, such combination inherits drawbacks from CPC, e.g. overshoots and slow reaction, and from HFS. Here, in particular, subscriber isolation is not removed that prevents background subscribers from yielding during overload.

3.5.6 Congestion Policing Queues

Description

The Congestion Policing Queue (CPQ) concept [155] is an academic proposal by the author of this thesis for implementing a resource allocation that is fair on a broader time scale. It can be seen as an advancement of CPC that broadens the meaning of congestion from the binary loss-focused understanding to an understanding using a continuous range based on queue size, i.e. which also captures queuing delays not resulting in a packet drop. In contrast to CPC, it combines resource allocation and policing in one element. CPQs only use local congestion information, i.e. the queue size of the internal queue, so a CPQ can only work if placed at the bottleneck. This also means that in contrast to CPC, CPQs reduce the congestion considered in making policing decisions from potentially all bottlenecks to the one bottleneck the CPQ is deployed at.

Discussion

The CPQ approach eliminates several drawbacks of CPC. CPQs use the internal queue size as congestion information, so there are no dependencies on end host support or auditing of external signaling. The queue size is also a more fine-grained and more frequent input information than the congestion information ConEx can provide: A meaningful value can be retrieved for every incoming packet. We showed that CPQs achieve the targeted displacement of heavy users' traffic during overload situations [155]. Nevertheless, CPQs are hard to tune properly.

Applying CPQs to the targeted use case requires deploying a CPQ instance at every interface carrying aggregated traffic, i.e. not only at every BNG downstream interface, but also at the downstream interfaces of the AGSs. CPQs perform better than CPC for the targeted use case with respect to several of our goals. The most important fact is that the shared queue immediately implies that background traffic can detect when bandwidth becomes scarce and therefore yields during overload.

Regarding some of the performance metrics, CPQs benefit from the properties of the hierarchical topology of our use case. This especially applies to queuing delay. Generally, due to the fixed buffer configuration of a CPQ, the average queuing delay of a user's traffic increases the fewer other users are active. Specifically, if the number of active users changes by a factor $1/N$, the average delay for the remaining users changes by the factor N. This does not create an issue since it only holds if the respective link is the bottleneck at that time. Due to the decreasing link speeds in the topology's hierarchy and the prevalence of CC, an aggregation interface in hierarchical access networks can only constitute the bottleneck if so many subscribers are active that their total capacity exceeds the aggregations link's capacity. The role of CC is crucial since CC ensures that the incoming traffic bandwidth does not significantly exceed the bottleneck capacity. So, when, e.g. during the night, only few subscribers are served by an aggregation link that can serve many more subscribers at full rate, no queue can build up at the aggregation interface. The same argument applies to the issue of increasing delay caused by putting several CPQ instances in sequence: if there is a significant queue at one level, the hierarchical topology makes sure that there is no significant queue at the other hierarchy levels. "Not significant" in this context means that traffic bursts may very well result in some packets buffered. But the queue will not grow to its limit, on the contrary, it will vanish within at maximum few RTTs.

There is the risk of TCP possibly causing a medium-term unfair rate allocation due to the shared queue and TCP-dominated traffic. As detailed in Section 2.4.2, Lautenschläger showed [106] that the throughput of TCP Cubic connections sharing a bottleneck with a shared queue varies by about a factor of three. Nevertheless, this only becomes an issue if the subscribers' access links do not serve as upper limit to the varying throughput, i.e. if the overload at the aggregation interface is huge. If the aggregation links are only slightly overloaded during peak periods, e.g. so that the subscribers receive 90 % of their maximum capacity, this also poses a limit on the possible unfairness. Therefore, the throughput of a single subscriber is expected to vary in the medium-term with this approach, but the extent of variation correlates with the extent of overload of aggregation links.

So, the approach of deploying CPQs at every interface on every aggregation level in a hierarchical access network may work sufficiently well for BE traffic in many scenarios. Yet, there are some issues. First, deployment of CPQ in an regional access network requires replacing all nodes in

one step, which is expensive and difficult in real networks. Second, in access networks there is not only BE but also priority traffic. To implement prioritization, a scheduler is necessary. So if CPQs were deployed at every node in the access network, a scheduler would also be required at every node. Moreover, each scheduler would need access to the policy repository.
We do not expect ISPs to be willing to fulfill these requirements because they contradict the ISPs' current architecture.

Summarizing, while CPQs could substantially improve the peak load situation with respect to our objectives, they have minor deficiencies. More important, due to their requirements, they are unlikely to be deployed.

3.6 Assumptions and Prerequisites

Some of the properties of the situation described in Section 3.2 are necessary for RADICCO to function properly or to achieve its objectives. Here, we list these properties and concisely discuss their role for RADICCO and their probability to change in the near future. This applies to technical properties as well as non-technical properties that relate to the human users. The technical properties regard the regional access network architecture (see Section 3.6.1), the hierarchical packet scheduler to be extended by RADICCO (see Section 3.6.2) and the way CCAs are used to differentiate foreground traffic from background traffic (see Section 3.6.3). Moreover, we discuss the relation to the users, i.e. their incentives to use background CCAs today and when RADICCO is deployed (see Section 3.6.4) and if subscribers should be informed about deployment of RADICCO (see Section 3.6.5).

3.6.1 Regional Access Network Properties

Bottleneck Mostly in the Access Network

As detailed in Section 2.1.2, in today's ISP networks the bottleneck is located in the regional access network for most of the traffic. A main reason for that is that current network design and planning is based on a fast but rather simple core network that particularly does not provide packet-level QoS. Allowing bottlenecks also in these parts of an ISP's network would require a fundamental redesign and expensive new devices. Moreover, there is no need for such functionality for a big share of traffic, e.g. all transit traffic that is switched through the network on physical layer.
Therefore, we assume ISPs' overall network architectures to focus bottlenecks to the access network also in the near future.

Oversubscription in Dimensioning Aggregation Links

RADICCO is only beneficial if aggregation links in hierarchical access networks are bottlenecks during peak periods, which can only happen if they are oversubscribed. As detailed in Section 2.1.1, oversubscription is part of any planning of packet switched networks for economic

reasons.
So, we expect this prerequisite being also met in the near future.

Hierarchical Scheduling and QoS Enforcement at the Edge

As detailed in Section 2.1.1, there are very good reasons to carry out scheduling and QoS enforcement not at all nodes in an access network but to define a service edge and execute these functions at the respective edge routers, i.e. BNGs. As long as the dimensioning of a hierarchical topology uses oversubscription, it makes no sense not to apply hierarchical scheduling, and thereby QoS enforcement, at the edge of that oversubscribed, hierarchical topology.
Since we expect that aggregation links will be oversubscribed, we expect that this prerequisite is also met in the near future.

Large Share of BE Traffic

The share of BE traffic is relevant for RADICCO for two reasons: First, since RADICCO operates on the BE traffic only, the less BE traffic there is, the less improvement RADICCO can achieve. Second, if shared links are used to transport all traffic classes of a subscriber, as it is today (and there is no reason for change), a smaller share of BE traffic means an increased share of priority traffic. This probably would result in higher volatility of the capacity available for BE traffic, i.e. available to RADICCO. Higher volatility in rates results in both background CCAs and RADICCO to function less reliably, so both the usage of background CCAs by the users and a deployment of RADICCO becomes less attractive.
Up to now, there are no signs that the share of priority traffic in residential broadband Internet traffic increases or that the set of prioritized services is extended. So, we expect that this prerequisite is also met in the near future.

Static Capacities of Aggregation Links

Today, capacities of aggregation links of wired access networks are static. Nevertheless, it may make sense to operate fiber-optical high speed links on lower than maximum transmission speeds, e.g. to save energy due to less complex modulation and Forward Error Correction (FEC) schemes. Though, the maximum speed, which is likely applied during peak load periods, is expected to be constant since the properties of the physical channel of the used fiber-optical links are static.
So, we expect that this prerequisite is also met in the near future.

Large Per-Subscriber Buffers

As we detailed in Section 2.4.4, foreground CCAs require rather large buffers at the bottleneck in order to fully utilize it. Due to the large existing deployment of hosts with such CCAs, ISP's are expected to continue configuring large buffers at interfaces and schedulers.
So, this prerequisite is expected to be also met in the near future.

Overload for Only Short Periods

As detailed in Section 2.1.3, aggregation links of today's residential access networks face overload only for short periods per day since most of the traffic for residential access networks belongs to services consumed by human users. We expect more traffic of non-interactive services in the future and thus an increasing amount of traffic that is rather independent of the time of day. Nevertheless, also interactive services will increase in traffic volume, e.g. due to higher resolutions used in VoD streaming.
In consequence, there is no reason to expect that non-interactive traffic will be the dominating share of traffic and thus we expect this prerequisite also being met in the near future.

3.6.2 Packet Scheduler

RADICCO does not depend on a particular scheduler, but the packet scheduler to be enhanced by RADICCO must meet some conditions in order to build a well performing system. A first set of properties is required for any scheduler at a downstream interface of a BNG of a hierarchical access network:

- Multiple hierarchies.
- Rate shaping.
- Prioritization (typically three or four classes).
- Adapting weights during operation.

The fourth requirement, support for weight adaptation, is not necessary for many access technologies. It must however be met for BNGs serving ADSL since ADSL allows for rate adaptation using Access Node Control Protocol (ANCP) signaling [RFC6320] due to changing channel conditions. Other access technologies such as VDSL usually do not support adapting the link capacity to changing physical conditions. Nevertheless, products intended for BNG deployment usually cover the whole range of potential scenarios to allow ISPs deploying a homogeneous architecture over heterogeneous access technologies. Therefore, today's routers with BNG functionality allow adapting the weights and maximum rates during ongoing operation, so their schedulers support adapting weights, too. For RADICCO, we do not require adapting maximum rates but only the weights. Nevertheless, both may be adapted. Our prototype implementation adapts both since weights and maximum rates are managed in one.

To be enhanced by RADICCO, the scheduler must additionally support

- Accessing a flow's current buffer usage.
- Sufficient performance to leave computation resource to execute RADICCO's calculations.

We will discuss both requirements in the following.

Accessing a Flow's Buffer Usage

The scheduler must access a data structure holding several variables of a flow during operation, e.g. the flow's weight but also the size of the head packet and if there is a packet in the queue. Moreover, the queue size, i.e. the number of bytes occupied in a flow's buffer, must be maintained as a state variable by the queuing discipline. We do not know commercial scheduler implementations but we assume that existing data structures accessible for the scheduler can be rather easily extended to include the existing state variable of the queue size.
So, we expect this requirement is met or can be met by all relevant scheduler implementations.

Performance

The design of RADICCO is based on the assumption that a scheduler is extended by RADICCO by integrating its algorithm into the scheduling algorithm rather than running it as a separate process. Therefore, we assume to require the necessary computational resources on the same Central Processing Unit (CPU) that the scheduler runs on. To estimate if a scheduler can be expected to leave sufficient computation resource to execute RADICCO's calculations, we, first, estimate the number of scheduling operations per time unit on a BNG downstream interface and their computational cost. Then, we estimate which complexity of RADICCO's operations is acceptable to be executed between two scheduling operations.

The number of scheduling operations per time unit depends on:

- The capacity of the BNG downstream interface.
- The average packet size.
- The minimum number of packets dequeued after each scheduling operation.

The interface's line rate typically is 1 Gbit/s or 10 Gbit/s today and in the near future can be expected to reach 40 Gbit/s or 100 Gbit/s. Regarding packet size distribution in the Internet there is only rather old data available [156], showing an average packet size of about 1000 byte. There is good reason to assume that average packet sizes in downstream have since increased because the bigger transmitted objects are, the more full-sized packets can be transmitted. The minimum number of packets dequeued after each scheduling operation is often assumed to be one, but this is not mandatory.
As an upper bound, we assume an average packet size of 1000 bytes, a 40 Gbit/s interface and single-packet dequeuing, i.e. 5 million scheduling operations per second.

The cost of a scheduling operation often depends on the number of flows, i.e. the product of

- The number of subscribers served by a BNG downstream interface.
- The number of QoS classes managed per subscriber.

As detailed in Section 2.1.1, we may assume several hundreds of subscribers and four QoS classes to be used, resulting in the number of flows being in the range of few thousands. This means, $log\ N$ can be safely bounded by $13 = log_2(8192)$. Since some algorithms such as WF^{2}Q+

require double-indexed data structures (see Section 2.2.4), such algorithms may be implemented with about thirty CPU operations per scheduling operation.
Here a remark on $O(1)$-DRR schedulers is necessary. To achieve *O(1)* complexity, the quantum for the smallest flow must be at least one maximum sized packet. Applying the RADICCO concept usually will result in an increased spread of flow weights since it reduces the weight of background flows to a value bg^{min} that is smaller than the smallest unmodified weight. If this increased spread of flow weights is to be handled with *O(1)*, the burstiness and therefore jitter will increase, and may reach unacceptable values. For example, assume the minimum weight corresponding to 1 Mbit/s, and the quantum being 1500 bytes. Consider a 100 Mbit/s subscriber, his flow will have a quantum of 150 KB or 1.2 Mbit. This is equivalent to 12 ms of continuous transmission at the access link's bandwidth. In contrast, at a 40 Gbit/s BNG interface this amount of data is transmitted within 0.03 ms. Therefore, a $O(1)$-DRR scheduler results in up to almost 12 ms of jitter for this 100 Mbit/s subscriber. So also for DRR scheduling, some additional effort may be recommended to provide better QoS in terms of jitter, e.g. by using Nested-DRR [157]. Nested-DRR has asymptotic constant complexity *O(1)*, but with a large constant.
Due to this behavior of true DRR schedulers and since N is not too large at BNG schedulers, we consider the computational effort of $O(logN)$ schedulers to be a reasonable upper bound estimate.

Regarding the processing units used in equipment deployed as BNGs there is no information publicly available. We assume their cores to perform a few thousand Million Instructions Per Second (MIPS), comparable to a slow general purpose CPU.

In consequence, schedulers with constant (*O(1)*) as well as logarithmic (*O(log N)*) complexity are expected to leave much of the computational resources of a one-core CPU unused, provided that the factors neglected in asymptotic analysis are not too big. If these resources are not used for other tasks in today's products, RADICCO could even be deployed to routers in the field via a software update. These resources might also be spent on tasks not related to scheduling, which therefore do not need to be executed on the same CPU as the scheduler. In that case, a next generation product could be equipped with RADICCO by adding more CPU cores without need to develop a faster CPU architecture.

So, we expect this requirement being met by many *O(1)* and *O(log N)* schedulers, and probably by the ones implemented in commercial products for BNG deployment.

3.6.3 Traffic Differentiation by Congestion Control

As described in Chapter 2.4, background CCAs react to delay while prevalent standard, i.e. foreground, CCAs react to loss / ECN only or they switch to such modes of operation if competing foreground traffic is detected. The approach of RADICCO assumes that this association of "background CCA" and "exclusively delay-controlled CCA" is bijective and stable.

There are good reasons for this assumption: First, any delay controlled connection will yield to a loss controlled one as long as the bottleneck buffer is sufficiently large, so traffic controlled by purely delay-controlled CCAs will be pushed to the background as soon as there is significant loss-controlled traffic. Small buffers could be an incentive for using delay controlled CCs for non-background traffic, but we expect large buffers to be configured in the near future as explained

above. Therefore, using a delay-controlled CCA must be interpreted as the source intending this traffic to yield to loss-controlled BE traffic, i.e. it must be considered as background traffic. Second, any delay-controlled connection will also yield to an ECN-controlled connection as long as the semantic of the ECN signaling remains as defined in [RFC3168] since then the reasoning given above applies to ECN, too. Third, even if recent activities in the research community to support low delay services by changing the semantic of ECN-CE and deploying dual AQMs in routers [154] will succeed, this low-delay foreground traffic still needs to be distinguished from legacy traffic, e.g. by re-interpreting the ECT1 signal. Thus, there will also exist a traffic class that does not implement the new ECN semantic but is handled as today's BE traffic class.
All in all, legacy loss-controlled (and ECN-controlled according to today's standards) traffic will continue to exist and will have to be treated like today. In this traffic class, delay-controlled CCAs can be used for identifying background traffic, so it is a stable association.

3.6.4 Incentives for Using Background Congestion Control Algorithms

Using background CCAs results in a disadvantage by their nature, namely receiving less than the fair share when competing with foreground traffic at a bottleneck. RADICCO further increases this drawback since it partly extends this behavior to bottlenecks in the aggregation network that today, with the deployed BNGs applying HFSs, are excepted from competition. Therefore, it is crucial to ensure sufficient incentives for the deciding parties to continue using such CCAs for non-urgent transmissions. This section will outline who shall be the addressee of such incentives, which incentives exist today and which options exist to provide further incentives for using background CCAs for transmitting non-urgent content.

Addressees of Incentives

At the downstream scheduler, only the sender's CCA is decisive for the traffic's behavior, not the subscriber's. Nevertheless, the receiver of a service usually has a big say in how a service is delivered, e.g. by how he requests that service. For example, many services can be retrieved by several ways, e.g. the ubuntu Linux distribution images can be retrieved either using a standard HTTP download, i.e. using a foreground CCA, or via BitTorrent, i.e. using uTP, a background CCA. This particular approach leaves the decision on how urgent that download is to the human user. For some other services, there is no human user in the loop, e.g. most OSs, depending on configuration, automatically download updates for themselves as well as for installed applications. This applies to current Windows, Linux, Android or Apple iOS OSs. For some applications and platforms, the application keeps itself up to date. For instance, software of the Mozilla foundation, such as the Firefox browser or the Thunderbird email client, manage the updating process on Windows OSs themselves. So, relevant addressees of incentives are users, OS designers and application programmers.

Of these three groups, end users are not the main target of incentives for BE differentiation for two reasons: First, the average user cannot be expected to understand and be aware of the technical consequences of his choices. Second, in many cases when a human user himself triggers a transmission, background priority is not the right choice since the user's QoE depends

on the achieved throughput, especially when the user in some way waits for completion of the transfer.

In contrast, for application and OS designers we can expect expertise and considerate design. This statement is proved by today's OSs: both Apple and Microsoft OSs use some kind of background or low priority rate control for their software update downloads. For Microsoft, there is BITS, which supports one foreground as well as several background priorities and aims to only use "idle network bandwidth" for background transfers [115]. Nevertheless, there is not much information on how BITS works to achieve this goal. Apple OSs use LEDBAT for software updates [114], probably including application updates. Unfortunately, there are strong indications, especially the public sources [125], that Apple uses the target delay recommended in the standard [RFC6817], which in many scenarios does not result in properly yielding to foreground traffic (see Section 2.4.3.5). Anyway, both examples show that OS designers are aware of the possibility of traffic differentiation by transport layer CC, their impact, and their responsibility. Moreover, OS designers have greater impact than application designers since today OSs are responsible for the by far largest amount of traffic for software updates.
Summarizing, OS designers and implementers are the most important players to be addressed by incentives to use CC for traffic differentiation.

Today's Incentive Situation

With today's classical hierarchical scheduler at the BNG, using CC traffic differentiation results in potential benefits only for users using the same access link.
CC traffic differentiation only makes a difference, when there are foreground and background transmissions competing for a shared bottleneck. This is only the case if the following conditions are met:

1. Both types of connections are active at the same time.
2. There is no other effective bottleneck for the foreground traffic.
3. They share a bottleneck that allows the background CCs to detect an increase in buffer usage caused by the foreground traffic.

This shared bottleneck typically is the access link or its corresponding scheduler node in the BNG's HFS. So, when applying an HFS at the BNG's downstream interface, increased throughput of foreground traffic is achieved at cost of the same subscriber's background traffic regardless of whether the BNG's scheduler limits the scheduled rate to the access link's capacity or to the subscriber's fair share of an congested aggregation link. Especially, no subscriber receiving foreground traffic can benefit at the cost of another subscriber receiving background traffic. Obviously, the potentially reduced throughput of background traffic is immediately compensated by the increased throughput of the concurrent foreground traffic. Since the increased foreground traffic bandwidth usually increases QoE, in today's situation there is a reasonable incentive for CC traffic differentiation as the wide-spread implementations proof.

RADICCO Impact on the Incentives System and Potential Adaptations

When applying RADICCO, the situation changes. Whenever the access link is the limiting bottleneck, the result is the same, there are no weights adapted. But if the bottleneck is an aggregation link, weights are adapted so that the throughput of background subscribers is reduced in favor of foreground subscribers. So, while CC-based traffic differentiation is only effective within the traffic of a subscriber today, i.e. with classical HFS, when applying RADICCO at the BNG, CC traffic differentiation becomes effective also among traffic of different subscribers. Inter-subscriber CC-based traffic differentiation shifts incentives, since now a subscriber can potentially benefit from other subscribers using background CCs, even without ever using background CCs.

There are several approaches to this challenge. The first option is to add counters to the scheduler implementation to enforce reciprocity: By checking these counters, RADICCO could allow a subscriber to benefit from other subscribers' background traffic only if and as much as this subscriber has received traffic detected as background traffic. Nevertheless, this approach further adds complexity and computational effort to the BNG's packet scheduler.
The second option is to use separate congestion policers as used in the CPQ approach (see Section 3.5.6). Both approaches provide long term incentives but might also result in short-term interference, impairing the subscriber's QoE.
The third approach is the simplest: do nothing. This approach is based on the assumption that the existing incentives will even gain importance in the future and will also be sufficient for the situation changed by deployment of RADICCO. Already today, most subscribers use several networked devices, including desktop computers, notebooks, tablet computers and smart phones, smart TVs, set-top-boxes and Internet radios. The variety as well as the number can safely be expected to continue to grow in the future, e.g. due recent trends like home automation or IoT. For the QoE of the human users, only few transmissions are relevant at a time. So, a higher number of active devices within one subscriber's domain increases theprobability of mutual interference if no measures are taken to prioritize the QoE relevant traffic. The only established option that does not require introducing additional signaling is traffic differentiation by CC. So, the incentive to yield to foreground traffic grows with increasing number of devices in the subscribers' networks. Moreover, as argued above most CC decisions are taken by OS designers, so are defined just once for many devices. The OS designers take decisions aiming to cope well with as many as possible scenarios, for instance with the access link being a bottleneck sometimes. Moreover, an end host usually cannot detect if there is a HFS in place or a RADICCO scheduler, and using background CC for background traffic does not inflict any drawbacks in case of HFS but comes with potential advantages.
So, there are many reasons that the existing incentive system is sufficient to work well also for RADICCO deployments and there are options to further increase incentives if necessary.

Summary on Incentive Situation

Although RADICCO fundamentally changes the incentive situation, we assume that probably there is no need for additional incentives besides the existing ones. If this assumption turns out to be wrong, there are still two options outlined above to adapt new deployments or even complement or upgrade existing ones.

3.6.5 Information of Subscribers

The question arises if and in which detail subscribers must or should be informed about the deployment of RADICCO. On the one hand, there are fields using other, less fair schedulers without informing the subscriber. For example, schedulers in cellular networks are very different from HFSs but there are also technical reasons for not treating all users the same in terms of bandwidth, especially the varying capacity of the device-specific channel. On the other hand, the vivid discussion on network neutrality may indicate that informing the subscribers could be advisable. Nevertheless, this is a non-technical question than may be answered differently from deployment to deployment. It is out of the scope of this work.

3.7 Algorithm Description

This section describes the algorithm of RADICCO, so it focuses on the hierarchical scheduler for regional access networks, i.e. the tree of schedulers at a BNG's downstream interface. Therefore, the following text refers to the graph of schedulers, not the network's graph. So, by "node" we refer to a scheduler instance in that tree representing an downstream interface in the hierarchical access network topology, but not to the network node as a whole, see also Figure 2.2.

In the following, the algorithm's core functions and attributes are presented, structured in six subsections: First, some necessary definitions and notations are introduced (see Section 3.7.1). Then an overview on how the algorithm executes the presented function blocks is given (see Section 3.7.2). Third, the algorithm for recognizing background traffic is described (see Section 3.7.4). Then the algorithm for calculating the leaf nodes' target rate for background traffic is presented (see Section 3.7.5). Fifth, the necessary state calculations of inner nodes, i.e. edge nodes and core nodes, are described (see Section 3.7.6) and finally, the algorithm for calculation of a node's effective rate is presented (see Section 3.7.7).

3.7.1 Definitions

For the description of the RADICCO algorithm, several terms and definitions are required.

Without loss of generality, we use rates instead of weights in all calculations. This is useful since the finally calculated effective rates may immediately serve as weights for schedulers that incorporate rate shaping as well as for schedulers that are used in combination with external rate shapers. External rate shapers do not need to be adapted: Their main purpose is preventing packet loss at the nodes of the access network and this goal is perfectly achieved even if a specific link cannot be fully utilized because of a bottleneck before that link. Nevertheless, the queue size at the rate shaper must be considered in traffic type recognition.

We must distinguish three significant types of hierarchy levels, i.e. types of nodes, in the hierarchical adaptation algorithm presented, yet there may be more levels in the hierarchy. The first type are the leaf nodes, representing schedulers for access links. The traffic of a leaf node is recognized as either foreground or background at a time. The second type are edge nodes, representing interfaces that aggregate traffic of multiple access links. So, edge nodes correspond

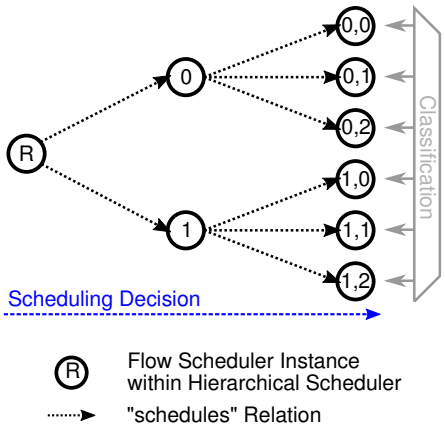


Figure 3.4: Corresponding hierarchical scheduler

to a feeding interface for an AN, thus carrying a mix of background and foreground traffic. The third type are core nodes that further aggregate aggregated traffic. In general, core nodes are all nodes but the ones on the two lowest levels in a hierarchical RADICCO scheduler. For a three-level hierarchy, e.g. as depicted in Figure 3.4, the only core node is the root node corresponding to the BNG's downstream interface.
We denote the root node by R, a core node by c, an edge node by e, an arbitrary inner node, i.e. any node except leaf nodes, by i and a leaf node by a tuple i, j where i identifies its parent and j its index at this edge node.

Generally, the algorithm works on different types of rates:

Capacity Rates C_x
A capacity rate directly represents the capacity of node x, the maximum rate of the corresponding interface in the hierarchical network topology. Today's HFS statically uses these rates for the respective nodes.

Target Rates $w_x(t)$ or $w_x^{FG}(t)$/$w_x^{BG}(t)$
A target rate of a node x defines a target, a guideline, for RADICCO. Often target rates are defined for the traffic share recognized as foreground traffic, indicated by superscript FG, or background traffic, indicated by superscript BG. The role of a target rate differs between background and foreground traffic due to the very goal of RADICCO, which is treating traffic differently depending on its traffic type.
Target rates of inactive nodes, i.e. nodes that do not have a packet ready to transmit, equal zero.
For foreground traffic, a target rate reflects the maximum possible rate that can be transported to the respective active receiver(s). RADICCO aims to finally assign that rate but during overload, this is not possible. So, a foreground target rate defines a maximum that often is not achieved.
In contrast, for background traffic, the target rates are usually below the maximum possible rate. For background traffic, the finally assigned rate may be greater or smaller than the target rate.
Except for foreground leaf nodes, a target rate is not constant over time.

Effective Rates $r_x(t)$

The effective rate of a node x represents the rate that the scheduling algorithm effectively uses for node x from now on until the next packet is dequeued from node x. The effective rate may be greater or smaller than the target rate but not bigger than the capacity rate.

The following notations will be used:

C_R The capacity of the scheduler's root node R, i.e. of the BNG's downstream link.

C_e The capacity of edge node e, i.e. of the downstream link from the aggregation switch to AN e.

$C_{i,j}$ The capacity of leaf node i,j, i.e. of the downstream link of the j's subscriber attached to AN i, subscriber i,j.

$w_i^{TT}(t)$ The target rate for the traffic type TT of inner node i at time t.

$w_{i,j}^{FG}$ The target rate of leaf node i,j if recognized as carrying foreground traffic. For all foreground leaf nodes holds at any time: $w_{i,j}^{FG} = C_{i,j}$.

$w_{i,j}^{BG}(t)$ the target rate of leaf node i,j at time t if recognized as carrying background traffic at time t.

$r_{i,j}^{FG}(t)$ The effective rate of leaf node i,j at time t if recognized as carrying foreground traffic at time t.

$r_{i,j}^{BG}(t)$ The effective rate of leaf node i,j at time t if recognized as carrying background traffic at that time t.

3.7.2 Execution Overview

A scheduler implementing RADICCO differs from a standard HFS in several ways. It keeps additional state at all nodes and the updates of these states are solely triggered by the execution of the leaf nodes' `pop()` and `enqueue()` functions. This section provides an overview on how the algorithms and functional modules presented in the following sections interact to achieve the goals aimed for.

First, RADICCO requires additional traffic monitoring to support traffic type recognition and to support the decision on where rates are adapted, but the necessary functions are simple and efficient. RADICCO keeps track of the load of any inner node, i.e. any aggregating interface. Moreover, RADICCO derives a binary load state, distinguishing peak load (*PeakLoadState*) from normal load (*NormalLoadState*). RADICCO does not depend on a specific detection mechanism. Finally, but most important, whenever RADICCO schedules a leaf node, i.e. a packet of its queue is dequeued, RADICCO derives the traffic type of this leaf based on its buffer usage (see Section 3.7.4). The recognized current type of traffic is maintained at each leaf node. To ease reading, we use the terms "background leaf node" and "foreground leaf node" referring to a leaf node whose traffic is recognized as the respective traffic type at that time. In

Algorithm 3.1: The adapted pop() function

```
1  Packet pop() {
2      if (isLeafNode) {
3          if (curPackets() == 1 || (inAdaptedOperationMode &&
4          (checkForTrafficTypeChange() || isBackground)))
5              updateAdaptation(null);
6      }
7      return super.pop();
8  }
```

Algorithm 3.2: The adapted enqueue() function

```
1  void enqueue(Packet p) {
2      if (isLeafNode && isEmpty) {
3          updateAdaptation(null);
4      }
5      super.enqueue(p);
6      return ;
7  }
```

addition, RADICCO knows two operation modes for a node, and every node in a hierarchical RADICCO scheduler is either in standard mode (*StandardOperationMode*) or rate adapting mode (*AdaptingOperationMode*). Only in the second mode of operation RADICCO potentially adapts rates of nodes also carrying background traffic.

Second, RADICCO adapts rates. The rate adaptation is only triggered by leaf nodes in *Adapting-OperationMode* on three occasions:

1. When a change in traffic type is detected
   by the call of `checkForTrafficTypeChange()` (see Algorithm 3.1).
2. When a background leaf node is scheduled, i.e. a packet is dequeued from its queue (see Algorithm 3.1).
3. When a leaf node of any traffic type changes its active state, i.e. a previously inactive leaf node receives a packet and becomes active or a previously active node dequeues its last packet and becomes inactive (see Algorithm 3.1 for dequeuing a leaf node's last packet and Algorithm 3.2 for enqueuing to an inactive leaf node).

To implement cases one and two, the `pop()` function is overwritten as shown in Java-like pseudo code in Algorithm 3.1. To implement case three, the `enqueue()` function is overwritten as shown in Algorithm 3.2.

The adaptation process itself is executed recursively as shown in Algorithm 3.3: First, necessary rate state variables are updated starting from the leaf, proceeding up to the root node. By carrying the resulting state change to the one parent node on the next level, the computational effort is independent of the number of nodes on that level. So, every rate adaptation only concerns the

Algorithm 3.3: The updateAdaptation() function

```
1  void updateAdaptation(StateChange d = null) {
2      StateChange delta = updateState(d);
3      if (!isRoot && inAdaptedOperationMode) {
4          getParent().updateAdaptation(delta);
5      }
6      updateEffectiveRate();
7  }
```

affected leaf node and all its ancestor nodes in *AdaptingOperationMode* up to the root node. So, at maximum exactly that many nodes are updated as the hierarchy has levels, typically three nodes. As shown in Algorithm 3.1, the rate adaptation is performed before calling the *pop()* function of the base scheduler. This ensures that the update of the scheduler's internal data structures (which depend on the scheduler in use) after dequeuing the packet from the leaf node's queue will already use the updated rates.

The calculations applied for both the state update and the update of the effective rate vary depending on the node type. For leaf nodes, only the target rate of background leaf nodes is updated (see Section 3.7.5). The algorithms for state update of inner nodes are presented in Section 3.7.6. Once the root is reached and thus the state update is completed, the assigned effective rates are re-calculated, this process starting at the root and proceeding down to the leaf. These calculations always follow a common concept and are described in Section 3.7.7 in a uniform representation.

3.7.3 Definition of Load States and Operating Modes

For any node in a hierarchical scheduler, RADICCO tracks via a binary variable if the respective interface is currently facing normal load or peak load. We refer to the two states as *NormalLoadState* and *PeakLoadState*, respectively. Due to its design, RADICCO is independent of the precise definition or detection algorithm. For the evaluated implementation, we identified peak load by the EWMA of the utilization being greater than 0.9.

RADICCO shall only alter the rate of links that are located behind a bottleneck at that time. Thus, RADICCO supports two modes of operation for all non-root nodes: *AdaptingOperationMode* and *StandardOperationMode*. Only in *AdaptingOperationMode*, RADICCO (potentially) modifies the rate of the respective node. This mode is only active, if a higher-level link possibly constitutes the bottleneck instead of the access link at that time. Consequently, a node is in *AdaptingOperationMode* if one of its ancestor nodes is in *PeakLoadState*, otherwise it is in *StandardOperationMode*. For a node in *StandardOperationMode*, as the name suggests, RADICCO does not alter the scheduler's operation. Since the root node has no parent, there cannot be any higher-level bottleneck. Therefore, at all times RADICCO aims to utilize its capacity to the maximum, which corresponds to the root node being in *StandardOperationMode*.

Accordingly, the detection and dependencies between these two modes are defined as follows:

- Any inner node's load is constantly monitored,
  independently of its load state or operation mode.
- Any child of a node in *PeakLoadState* is in *AdaptingOperationMode*.
- Any child of a node in *AdaptingOperationMode* is in *AdaptingOperationMode*.
- Any child of a node in *NormalLoadState* and *StandardOperationMode* is in *StandardOperationMode*.

Therefore, a change of operation mode often concerns only partial subtrees, minimizing the number of operation mode switches. We highlight this by an example: Assume the whole scheduler tree being in *NormalLoadState*. This directly implies that all nodes are in *Standard-OperationMode*. When the root node *R* enters *PeakLoadState*, all its child nodes with their subtrees switch to *AdaptingOperationMode*. While *R* is in *PeakLoadState*, the load of the inner nodes in the subtrees is still monitored, so e.g. if a child *i* detects peak load, *i* enters *PeakLoadState*. Therefore, when node *R* leaves *PeakLoadState*, all nodes in its subtree switch to *StandardOperationMode* except the child nodes of node *i* and their subtrees, which remain in *AdaptingOperationMode*.

3.7.4 Traffic Type Recognition

In order to identify the dominating behavior of a subscriber's aggregate, i.e. of a flow in terms of packet scheduling, the CCAs' behaviors and assumed topological properties are exploited.
For foreground CCAs, we know that they fill the buffer at the bottleneck and frequently cause packet losses there. For background CCAs, we know that they do not use a significant amount of buffer space at the bottleneck. Both types of CCAs may use the full bandwidth of the bottleneck. For residential Internet traffic, the bottleneck is usually located within the delivering regional access network, i.e. within the HFS. In that case, monitoring the development of a flow's queue size or the flow's packet drops is sufficient to recognize the dominating traffic type of that flow, i.e. the respective traffic aggregate. Although this holds for all hierarchy levels, the potential gain is obviously the bigger the lower the hierarchy level RADICCO operates on. Therefore, RADICCO only carries out traffic type recognition on per-subscriber level, the smallest granularity scheduled at BNGs.

For the technical realization, we need to observe the respective per-subscriber queues for some time. When designing our recognition algorithm, we only considered options that require, first, only low overhead and, second, a rather short observation time. With these constraints, we found that the most reliable approach to recognize the traffic type is to check if the buffer usage has been below a threshold for at least some defined time interval. Details on reasoning and alternatives are given in Section 3.8.5.

Due to the AIMD behavior of foreground CCAs, the threshold must be scaled with the BDP for nodes of different rates, i.e. with the node's rate. So, although the monitoring is performed using a specific absolute threshold $tt\_recog_{i,j}^{absQThresh}$ for every leaf node i, j, RADICCO provides a global configuration parameter that defines a relative threshold $tt\_recog^{relQThresh}$ giving the threshold as a fraction of the leaf node's buffer size.

In calibration simulations, we found that a relative threshold $tt\_recog^{relQThresh} = 0.35$ requires only an observation time of $tt\_recog^{obsDuration} = 2\,\mathrm{s}$ in order to achieve close to zero false positives. Nevertheless, the reliability of any traffic type recognition algorithm also depends on the variability of the overall system. For instance, many simulations presented in Chapter 4 resulted in worse traffic type recognition than seen in our calibration simulations. Developing a product from RADICCO would require further tuning of $tt\_recog^{relQThresh}$ and $tt\_recog^{obsDuration}$ and calibrating them with real traffic.

3.7.4.1 Initial Traffic Type

At some point in time, a subscriber connects to its access node for the first time, or this access node or even the BNG is restarted or replaced. Whatever the default traffic type in RADICCO after startup is, its impact is negligible: after a short period, any foreground traffic should have been recognized as such, and same applies for background. So, this is not the concern of this section. Today, with flat rates being the by far prevalent plan for Internet services, subscribers are usually not active all the time they are connected to their ISP, so we can decide with which traffic type a subscriber shall be recognized initially after such idle period. The best choice heavily depends on the transported traffic, especially the probability of traffic type changes. So, the initial traffic type for products implementing RADICCO should be left for configuration (other options are discussed in Section 3.8.6).

For our implementation, we chose to ignore idle times. This in consequence means that an idle subscriber will be recognized as background subscriber latest after $tt\_recog^{obsDuration}$ minus the time for depleting a queue of $tt\_recog_{i,j}^{absQThresh}$ as described in Equation 3.1.

$$idleToBackground_{MAX} = tt\_recog^{obsDuration} - \frac{tt\_recog_{i,j}^{absQThresh}}{r_{i,j}^{FG}(t)}. \tag{3.1}$$

So, $idleToBackground_{MAX}$ is just some tens of milliseconds less than the observation duration used by the traffic type detection, i.e. just a bit less than two seconds.

We argue that often the traffic type does not change during short idle intervals and that in general there is little harm caused by recognizing traffic as background by default. The reason is that the slow start performed by foreground CCAs fills the buffer fast, causes the buffer usage to exceed the threshold and by that the traffic to be correctly recognized as foreground traffic. This low impact does not only apply to newly starting foreground connections but also to foreground connections having been idle for more than their RTO, which is for today's common RTTs often the one second minimum value. The duration of this type change of course depends on the minimum background target rate $bg_{i,j}^{min}$ and the absolute size of the buffer usage threshold $tt\_recog_{i,j}^{absQThresh}$. We also expect the importance of idle times for RADICCO's traffic type recognition to decrease in the near future. We expect that with the increasing number of always-on networked devices and with IoT being used also by the average subscriber, the number of low rate but persistent connections, e.g. for status updates, will increase substantially. Such connections often use TCP and transmit small keep-alive messages in medium intervals, typically in the range of one to two minutes. For instance, connections of email push services using Internet Message Access Protocol (IMAP) IDLE (defined in [RFC2177]) are kept open for a long time but are often idle for many minutes except short keep-alive messages at intervals

(Linux default is 75 seconds) that carry no data. With the number of such connections increasing substantially, absolute idle periods become shorter and do not provide information on user-relevant, i.e. QoE-relevant, transfers any more.
Summarizing, we consider the initial traffic type being a configuration parameter and ignored idle durations in our implementation.

3.7.5 State Calculation for Leaf Nodes, i.e. Calculation of Background Target Rates

For leaf nodes, updating the target rate of a leaf node currently recognized as background and in *AdaptingOperationMode* is the only calculation that is performed in `updateState()` function. A phase of target rate adaptation starts as soon as both criteria are met. This phase ends and the background target rate is reset to the node's capacity $C_{i,j}$ when one of the two conditions is found to be not met anymore. This is only checked when the leaf node is scheduled. When a background leaf node in *AdaptingOperationMode* becomes inactive because its last packet is dequeued, the target rate is stored in a temporary variable and the background target rate is set to zero. When a background leaf node in *AdaptingOperationMode* becomes active again, the old background target rate is restored.

While the target rate of a subscriber i,j recognized as receiving foreground traffic is always equivalent to its access link's capacity, i.e. $w_{i,j}^{FG} = C_{i,j}$, when recognized as receiving background traffic and in *AdaptingOperationMode* the subscriber's target rate is reduced, so $w_{i,j}^{BG}(t) \leq C_{i,j}$.

The target rate $w_{i,j}^{BG}(t)$ must be slowly reduced to avoid false foreground recognition and under-utilization due to overreaction by the sender's CCA.
The root cause for a false foreground recognition is that the sender cannot become aware of a rate reduction earlier than one RTT after the change became effective at scheduler. Therefore, the sender continues to send packets according to the state before the change for one RTT. During this RTT, packets will usually arrive faster at the BNG than they are dequeued, i.e. the subscriber's queue grows. If a background subscriber's rate is reduced too fast, the queue size exceeds the threshold $tt\_recog_{i,j}^{absQThresh}$, the subscriber is recognized as foreground traffic and therefore his target rate is set to $C_{i,j}$. The background CCA will not keep the buffer usage that high, so after a while, this subscriber is again recognized as background subscriber, restarting the cycle.
Moreover, a too fast rate reduction of background subscribers may result in the respective CCA to overly reduce its cwnd, possibly to the lowest possible value, which is a zero cwnd for some background CCA, e.g. uTP. So, also the sender's CCA may introduce a unstable, oscillating behavior.
Any such oscillating patterns should be avoided since in the worst-case they might escalate and result in also fast changing capacities available to the foreground traffic. This would provoke an unacceptable service for the foreground traffic.

Therefore, RADICCO is designed to reduce the target rate of a background leaf node on every dequeue event just by a constant, rather small amount β_{reduce}, but not below a predefined

minimum $bg_{i,j}^{min}$. Equation 3.2 shows the calculation for the target rate when the n-th packet since the begin of this phase of target rate adaptation, p_n, is scheduled.

$$w_{i,j}^{BG}(p_n) = max(bg_{i,j}^{min}, C_{i,j} - n * \beta_{reduce}) \tag{3.2}$$

$$= max(bg_{i,j}^{min}, w_{i,j}^{BG}(p_{n-1}) - \beta_{reduce}) \tag{3.3}$$

The calculation executed on each call of `updateState()` is shown in Equation 3.3. This target rate adaptation scheme results in a decreasing speed of rate reduction over time, if the effective rate equals the target rate. This is not necessarily the case since RADICCO fills up background rates if the capacity cannot be utilized by foreground traffic (see Section 3.7.7). If the effective rate depends on the target rate, the advantage of this mechanism is that each potential input sample of the sender's control loop reflects the same change in the bandwidth domain. We argue that a subscriber should always receive a minimum service to avoid starving background connections and moreover, to protect certain rate-limited services such as VoIP (see Section 3.8.5.3).

Both parameters, β_{reduce} and $bg_{i,j}^{min}$, can be adapted to the deployment scenario. While β_{reduce} depends on the traffic's background CCA and thus should be globally adapted to the deployment scenario, the minimum target rate $bg_{i,j}^{min}$ can be a per-subscriber configuration parameter.
For the choice of $bg_{i,j}^{min}$, the deployment scenario is crucial: On the one hand, if RADICCO shall be used to gain a large amount of bandwidth from background traffic to tolerate substantially higher offered load than the aggregation bottlenecks can handle, low $bg_{i,j}^{min}$ should be configured to allow a rather large gain by RADICCO. On the other hand, if RADICCO is only meant to absorb peaks of load hardly exceeding the capacities of the aggregation links, rather high values $bg_{i,j}^{min}$ should be configured to reduce the impact of foreground traffic falsely recognized as background traffic. The expected share of background traffic must also be taken into account.
Similar considerations apply to β_{reduce}: If there is only little gain necessary and background transmissions are known to mostly last long, a lower value should be configured to improve smoothness of operation.

In our calibration simulations using uTP and TCP Vegas as background CCAs and RTTs of up to 100 ms, we found a nominal decrease of $\beta_{reduce} = 37.5\ \frac{kbit}{s*packet}$ to ensure stable operation and fast rate reduction at the same time. In the evaluation simulations, we use $bg_{i,j}^{min} = 1Mbit/s$ for all leaf nodes, equaling 1/20 of the access link speed $C_{i,j} = 20Mbit/s$. We choose this rather low value to evaluate the stability of the overall system and to show the extent of potential disadvantages on otherwise rate-limited traffic. For deployment, we rather expect $bg_{i,j}^{min}$ to be set higher, maybe to relative values of the subscriber's capacity $C_{i,j}$, e.g. 50 % of $C_{i,j}$.

3.7.6 State Calculations for Inner Nodes

This section describes the state maintained by the `updateState(StateChange d)` function for inner nodes. For easier reading, we describe all state variables by closed formulas in the following, although the implementation works based on the state change at the lower level node that called this function.

Every inner node in general carries foreground as well as background traffic, and due to smaller capacities at nodes further down the tree there also exist upper limits for both types of traffic.

When describing the algorithm, we will need further definitions. First we will define the notations that will be used, then we will provide the detailed definitions.

$sw_i^{FG}(t)$	The sum of target rates for foreground traffic of all child nodes of inner node i.
$sw_i^{BG}(t)$	The sum of target rates for background traffic of all child nodes of inner node i.
$o_i(t)$	The local load factor of inner node i.
$w_i^{FG}(t)$	The target rate of foreground traffic at inner node i.
$w_i^{BG}(t)$	The target rate of background traffic at inner node i.
$r_{MAX}{}_i^{BG}(t)$	The maximum possible rate of background traffic at inner node i due to restrictions of links transporting the background traffic further down the hierarchy.
$sr_{MAX}{}_i^{BG}(t)$	Tthe sum of maximum possible rates of background traffic of all child nodes at inner node i.
$\bar{h}_i^{BG}(t)$	The best-case headroom for background traffic at inner node i.
$r_i^{FG}(t)$	The effective rate of foreground traffic at inner node i.
$r_i^{BG}(t)$	The effective rate of background traffic at inner node i.
$r_i(t)$	The effective rate of inner node i. Calculated as $r_i(t) = r_i^{FG}(t) + r_i^{BG}(t)$.

Sum of Target Rates for Foreground Traffic of All Child Nodes at Inner Nodes

Regarding the sum of target rates for foreground traffic of all child nodes of an inner node, we must distinguish between edge and core nodes.
In Equation 3.4 the formula for an edge nodes e is given. In that case only child nodes recognized as foreground leaves are considered in the sum.

$$sw_e^{FG}(t) = \sum_{e\text{'s FG child nodes } j} w_{e,j}^{FG} \tag{3.4}$$

Equation 3.5 shows the formula for a core node c, where all child nodes are inner nodes and in general carry both types of traffic.

$$sw_c^{FG}(t) = \sum_{c\text{'s child nodes } j} w_j^{FG} \tag{3.5}$$

Sum of Target Rates for Background Traffic of All Child Nodes at Inner Nodes

Analog to the foreground rates, there are also two formulas for the sum of target rates for background traffic of all child nodes at inner nodes. Equation 3.6 gives the formula for an edge node e, Equation 3.7 for a core node c.

$$sw_e^{BG}(t) = \sum_{e\text{'s BG child nodes } j} w_{e,j}^{BG} \tag{3.6}$$

$$sw_c^{BG}(t) = \sum_{c\text{'s child nodes } j} w_j^{BG} \tag{3.7}$$

Local Load Factor of an Inner Node

The local load factor $o_i(t)$ of an inner node i captures the ratio between the target rates of its child nodes and its own nominal capacity C_i as shown in Equation 3.8.

$$o_i(t) = \frac{sw_i^{FG}(t) + sw_i^{BG}(t)}{C_i} \tag{3.8}$$

Note that the nominal capacity is not necessarily available at that time. At the same time, if i is a core node this load factor already considers limitations in the subtrees of the child nodes by referring to $sw_i^{FG}(t)$ and $sw_i^{BG}(t)$.
So, this load factor does not reflect the overall view but the local view taking into account the whole subtree below i. In consequence, if $o_i(t) > 1$, node i is definitively overloaded. If $o_i(t) \leq 1$, it may be not overloaded or may be overloaded because it receives a lower effective rate $r_i(t)$ than its capacity C_i.

Target rate of Foreground and Background Traffic at Inner Nodes

The target rates of foreground traffic at an inner node i, $w_i^{FG}(t)$, and of background traffic at an inner node i, $w_i^{BG}(t)$, are calculated analogously, so we do not make a further difference here. $w_i^{FG}(t)$ is either the sum of foreground target rates of all child nodes of node i, if there is no local overload at i, i.e. $o_i(t) \leq 1$. Or, if there is local overload at node i, the target rate is the sum described above divided by node i's local overload factor $o_i(t)$. Equation 3.9 shows that definition.

$$w_i^{FG}(t) = \begin{cases} sw_i^{FG}(t) & \text{if } o_i(t) \leq 1 \\ \frac{sw_i^{FG}(t)}{o_i(t)} & \text{if } o_i(t) > 1 \end{cases} \tag{3.9}$$

$w_i^{BG}(t)$, is defined analogously to $w_i^{FG}(t)$ as shown in Equation 3.10.

$$w_i^{BG}(t) = \begin{cases} sw_i^{BG}(t) & \text{if } o_i(t) \leq 1 \\ \frac{sw_i^{BG}(t)}{o_i(t)} & \text{if } o_i(t) > 1 \end{cases} \tag{3.10}$$

Maximum Rate of Background Traffic at Inner Nodes

The maximum possible rate of background traffic at an inner node i, ${r_{MAX}}_i^{BG}(t)$, is defined as the minimum of, first, the maximum background traffic the child nodes could accept and, second, the maximum bandwidth available for background traffic at that inner node i. The maximum bandwidth available for background traffic at an inner node i is given by the node's capacity C_i less the bandwidth that shall be allocated to foreground traffic, $w_i^{FG}(t)$. Regarding the maximum background traffic the child nodes could accept, we again have to distinguish between edge nodes and core nodes since there is no common variable we can resort to. Equation 3.11 shows the formula for an edge node e, and the formula for a core node c is given in Equation 3.12.

$$ {r_{MAX}}_e^{BG}(t) = min\left(\sum_{e\text{'s BG child nodes } j} C_{e,j} \; , \; C_e - w_e^{FG}(t) \right) \tag{3.11} $$

$$ {r_{MAX}}_c^{BG}(t) = min\left(\sum_{c\text{'s child nodes } i} {r_{MAX}}_i^{BG}(t) \; , \; C_c - w_c^{FG}(t) \right) \tag{3.12} $$

Sum of Maximum Rates of Background Traffic of All Child Nodes at Inner Nodes

To calculate the sum of maximum rates of background traffic of all child nodes at an inner node i we again have to distinguish between edge nodes and core nodes. In Equation 3.13 the formula for an edge node e is given. Here all background child nodes e, j are considered with their capacity $C_{e,j}$. Equation 3.14 shows the formula for a core node c.

$$ {sr_{MAX}}_e^{BG}(t) = \sum_{e\text{'s BG child nodes } j} C_{e,j} \tag{3.13} $$

$$ {sr_{MAX}}_c^{BG}(t) = \sum_{c\text{'s child nodes } j} {r_{MAX}}_j^{BG}(t) \tag{3.14} $$

Best-Case Headroom for Background Traffic at Inner Nodes

The best-case headroom for background traffic at a node n, $\bar{h}_n^{BG}(t)$, describes the amount of bandwidth that could be allocated to background traffic additionally to its target rate $w_n^{BG}(t)$ if node n was assigned its full capacity, i.e. $r_n(t) = C_n$ which is the best-case effective rate.

We define the calculation first for a leaf node i, j, where the calculation is trivial as seen in Equation 3.15.

$$ \bar{h}_{i,j}^{BG}(t) = C_{i,j} - w_{i,j}^{BG}(t) \tag{3.15} $$

For an inner node i, the maximum rate of possibly transported background traffic is ${r_{MAX}}_i^{BG}(t)$. This allows to precisely define $\bar{h}_n^{BG}(t)$ as the minimum of i's capacity less the sum of all target rates of its child nodes and the maximum rate of background traffic at i less the sum of target rates for background traffic of all child nodes of i, but no less than zero. Using the already defined variables, we give a common equation for edge and core nodes in Equation 3.16.

$$\bar{h}_i^{BG}(t) = max\left(0\ ,\ min\left((C_i - sw_i^{FG}(t) - sw_i^{BG}(t)\ ,\ r_{MAX}{}_i^{BG}(t) - sw_i^{BG}(t)\right)\right) \tag{3.16}$$

3.7.7 Calculation of Effective Rates

When calculating effective rates from target rates, there are basically two models possible: The strict approach is to simply assign the calculated target rates as effective rate, if available capacity allows, and trim all rates fairly if not. The utilization-maximizing approach is to always aim at utilizing available bandwidth to the maximum and therefore fill up rates if and where possible. There are good reasons for the utilization-maximizing approach and therefore to use background target rates as a guidance only (see Section 3.8.3). Therefore, we chose this approach when designing RADICCO.

Updating the effective rates is the second phase of the `updateAdaptation()` function and starts at the root node. Whenever the following calculations are executed, the state variables, which characterize the demand of child nodes for the two traffic types, have just been updated. In addition, the available rate at the node we start with is known since the root node always can make use of its full capacity. The algorithm starts with calculating the effective rate of the root node and then the recursive call calculates all effective rates to the leaf node that triggered the state update.
We describe this algorithm by first providing the formula for the effective rate at the root node. We then provide the calculations for the effective rates of background and foreground traffic of all child nodes for an inner node given its effective rate. This allows to recursively calculate the effective rates of all nodes.

Calculating the Effective Rate of the Root Node

The effective rate of the root node R is only less than its capacity C_R if the child nodes cannot handle that much traffic. This is the case exactly if the sum of maximum rates of foreground and background traffic of all child nodes is less than the capacity C_R, in which case this sum also is the effective rate of R. For foreground traffic, the sum of target rates of all child nodes corresponds to the maximum rate. Therefore, $r_R(t)$ is calculated as given in Equation 3.17.

$$r_R(t) = min\left(C_R\ ,\ sw_R^{FG}(t) + r_{MAX}{}_R^{BG}(t)\right) \tag{3.17}$$

Calculating the Effective Rates of Child Nodes

For the calculation of effective rates of the child nodes of an inner node i, we distinguish three load levels of that inner node i: underload, light overload and heavy overload. These load levels just represent the three possible cases in the utilization-maximizing approach: Either we can give maximum top-up for all child nodes, or we can at least assign more than requested although not the maximum, or there even is need to trim, see also Figure 3.5. For each load level, two steps of calculation are needed:

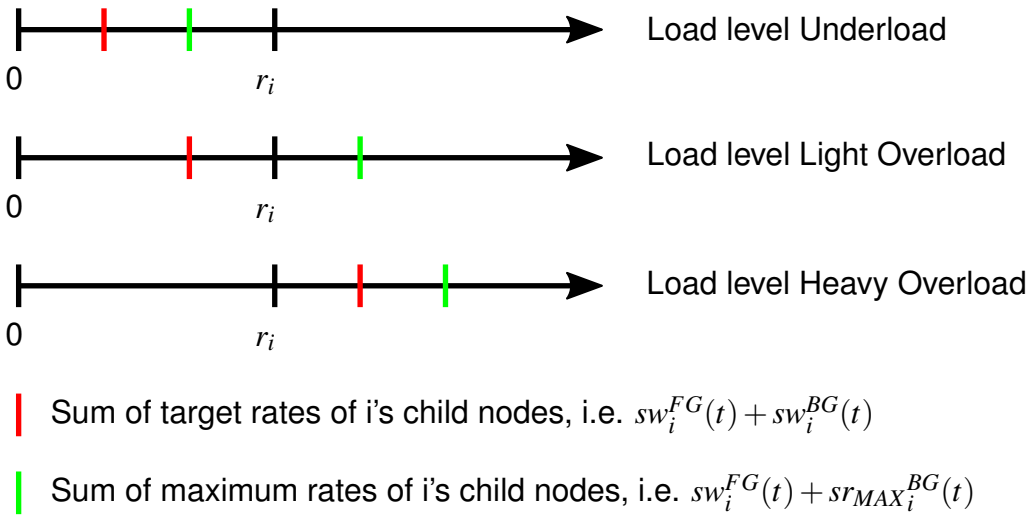


Figure 3.5: Visualization of load levels by relation between decisive state variables (not to scale)

1. Calculate the effective rate to be allocated to foreground and background traffic at node i.
2. Calculate for each child node j its respective share in both foreground and background.

Strictly, step one is only needed for the root node since for all other hierarchy levels the respective value have already been calculated by the parent node.

Underload, i.e. $sw_i^{FG}(t) + sr_{MAX}{}_i^{BG}(t) \leq r_i(t)$

This load level is characterized by inner node i being able to transport the full rates that the lower nodes can handle, i.e. the sum of maximum rates of foreground and background traffic is smaller than the node's capacity. This case exists in *AdaptingOperationMode* only because RADICCO enters *AdaptingOperationMode* based on peak load detection, i.e. there is not necessarily overload.
As result, for node i effective background and foreground rates are equivalent to their maximum rates as shown in Equations 3.18 and 3.19.

$$r_i^{FG}(t) = sw_i^{FG}(t) \tag{3.18}$$
$$r_i^{BG}(t) = r_{MAX}{}_i^{BG}(t) \tag{3.19}$$

For all child nodes, the effective rate is equivalent to the maximum foreground and background traffic it can accept. The calculation depends on the type of child node due to the difference between inner nodes and leaf nodes. If node i is a core node, child node j is an inner node, too. For that case, the maximum foreground and background traffic is equivalent to the already calculated target rate of foreground traffic and maximum rate of background traffic as shown in Equations 3.20 and 3.21.

$$r_j^{FG}(t) = w_j^{FG}(t) \tag{3.20}$$
$$r_j^{BG}(t) = r_{MAX}{}_j^{BG}(t) \tag{3.21}$$

If node i is an edge node, its child nodes (i, j) are leaf nodes. In that case, the maximum bandwidth is equivalent to the capacity $C_{i,j}$ and is completely assigned to the current traffic type of that node. This is illustrated in Equations 3.22 and 3.23.

$$r_{i,j}^{FG}(t) = \begin{cases} C_{i,j} & \text{if } (i,j) \text{ is recognized as foreground at time } t \\ 0 & \text{if } (i,j) \text{ is recognized as background at time } t \end{cases} \tag{3.22}$$

$$r_{i,j}^{BG}(t) = \begin{cases} 0 & \text{if } (i,j) \text{ is recognized as foreground at time } t \\ C_{i,j} & \text{if } (i,j) \text{ is recognized as background at time } t \end{cases} \tag{3.23}$$

Light Overload, i.e. $sw_i^{FG}(t) + sw_i^{BG}(t) < r_i(t) < sw_i^{FG}(t) + {sr_{MAX}}_i^{BG}(t)$

This load level is characterized by the inner node being able to serve all active child nodes with at least their target rate, but not with their maximum rates. This concerns background traffic only since for foreground the maximum rates are equal to their target rates by definition.

Therefore, effective foreground rate of an inner node i is equal to the sum of target rates for foreground traffic of all its child nodes, i.e. $sw_i^{FG}(t)$ as shown in Equation 3.24 .

$$r_i^{FG}(t) = sw_i^{FG}(t) \tag{3.24}$$

Regarding background traffic at this inner node there is some headroom allowing to assign some child nodes' background traffic higher effective rates than their target background rates. Here we need to calculate the effective background headroom, which is quite similar to the best case background headroom but is based on the now available effective rate $r_i(t)$ instead of the capacity C_i. We formally define the effective background headroom $h_i^{BG}(t)$ of an inner node i in Equation 3.25.

$$h_i^{BG}(t) = max\left(0\ ,\ min\left(r_i(t) - sw_i^{FG}(t) - sw_i^{BG}(t)\ ,\ {r_{MAX}}_i^{BG}(t) - sw_i^{BG}(t)\right)\right) \tag{3.25}$$

This available headroom is fairly distributed among all potential recipients. When distributing this headroom available at i among the child nodes, we take into account the limits applicable at each child node, i.e. its best-case headroom for background traffic. We chose to distribute the headroom proportionally fair (relative to the headroom) amongst all child nodes, i.e. RADICCO increases the effective rate of background traffic of every child node by adding the same fraction of the respective child node's best-case headroom to the child node's target rate. Therefore, we calculate i's fill up fraction $f_i(t)$ as depicted in Equation 3.26

$$f_i(t) = \frac{h_i^{BG}(t)}{\sum_{i\text{'s child nodes } j} \bar{h}_j^{BG}(t)} \tag{3.26}$$

The effective rates of inner node i's child nodes j are calculated as follows:

The effective foreground rate $r_j^{FG}(t)$ is equivalent to j's target rate for foreground traffic, as depicted in Equation 3.27.

The effective background rate $r_j^{BG}(t)$ is equivalent to j's background target rate plus i's fill up fraction $f_i(t)$ times j's best-case headroom for background traffic $\bar{h}_j(t)$ as shown in Equation 3.28.

$$r_j^{FG}(t) = w_j^{FG}(t) \tag{3.27}$$

$$r_j^{BG}(t) = w_j^{BG}(t) + f_i(t) * \bar{h}_j(t) \tag{3.28}$$

Heavy Overload, i.e. $r_i(t) \leq sw_i^{FG}(t) + sw_i^{BG}(t)$

This load level is characterized by the sum of the child nodes' target rates exceeding the considered inner node i's effective rate, therefore all child node rates are shortened by i's effective load factor $O_i(t)$. In contrast to the local load factor $o_i(t)$, the effective load factor $O_i(t)$ is based on i's effective rate $r_i(t)$ instead of its constant capacity C_i. $O_i(t)$ is consequently defined as shown in Equation 3.29.

$$O_i(t) = \frac{sw_i^{FG}(t) + sw_i^{BG}(t)}{r_i(t)} \tag{3.29}$$

So, the effective rates of background and foreground traffic at node i are equivalent to the respective sum of target rates of all child nodes shortened by the overload factor as shown in Equation 3.30 and 3.31.

$$r_i^{FG}(t) = \frac{sw_i^{FG}(t)}{O_i(t)} \tag{3.30}$$

$$r_i^{BG}(t) = \frac{sw_i^{BG}(t)}{O_i(t)} \tag{3.31}$$

The respective rates for a child node j are equivalent to its respective target rate shortened by node i's load factor. Again, due to the difference in the child nodes' behavior, we give two sets of definitions. Equations 3.32 and 3.33 present the formulas for a leaf node (e, j), child of edge node e, and Equations 3.34 and 3.35 show the calculations for an inner node k, child of a core node c, each for foreground and background traffic respectively.

$$r_{e,j}^{FG}(t) = \begin{cases} \frac{w_{e,j}^{FG}(t)}{O_e(t)} & \text{if } (e,j) \text{ is recognized as foreground at time } t \\ 0 & \text{if } (e,j) \text{ is recognized as background at time } t \end{cases} \tag{3.32}$$

$$r_{e,j}^{BG}(t) = \begin{cases} 0 & \text{if } (e,j) \text{ is recognized as foreground at time } t \\ \frac{w_{e,j}^{BG}(t)}{O_e(t)} & \text{if } (e,j) \text{ is recognized as background at time } t \end{cases} \tag{3.33}$$

$$r_k^{FG}(t) = \frac{w_k^{FG}(t)}{O_c(t)} \tag{3.34}$$

$$r_k^{BG}(t) = \frac{w_k^{BG}(t)}{O_c(t)} \tag{3.35}$$

3.8 Rationales for Core Design Decisions

In this section, we present the core rationales for important design decisions. The first two subsections on the granularity of operation (Section 3.8.1) and on our strategy for state updates (Section 3.8.2) cover topics that are, first, rather independent of all other design decisions and, second, are rather straight forward. In the third subsection, we discuss our decision to fill up background rates (Section 3.8.3), which also is a fundamental design decision but the decision is less obvious. The last three subsections are about how RADICCO identifies background traffic

(Section 3.8.5), how RADICCO calculates target rates for background leaf nodes (Section 3.8.4) and which traffic type is assumed after idle phases (see Section 3.8.6). These three topics are closely related and each respective reasoning is related to the decisions taken in the two other areas.

3.8.1 Granularity of Operation

We decided to design RADICCO to completely operate on subscriber or access link level, i.e. to detect the traffic type for the aggregate of a subscriber's access link and to adapt weights starting on access link level. As we will explain, a design operating on transport layer connections would be more expensive and does not make sense, and a design operating on AN level would perform much worse.

A transport layer connection is the next smaller easily detectable unit of communication compared to the traffic aggregate of a subscriber. Operating on transport layer connections may seem natural since each connection is controlled by one CCA instance at the respective sender, while an access link carries a mix with regard to CC. Nevertheless, the behavior of a CCA can only be observed if it receives congestion feedback, so this approach requires to schedule either each connection or at least background and foreground traffic separately. In turn, this requires the scheduler to determine how the current flows or the two traffic types share the subscriber's access link capacity. This means disabling the prioritization capability of using different CCAs and replacing it by an ISP policy defining rate allocations on connection level. First, this violates the end-to-end principle unnecessarily and maybe even affect network neutrality. Second, recognizing the traffic type of every single connection is more difficult than recognizing the traffic type of an aggregate and thus will result in more false recognitions. Third, the necessary allocation policies cannot correctly reflect the intentions of the subscriber or the senders.
Moreover, as we detailed in Section 2.4.2, a connection controlled by foreground CCA usually dominates the behavior of the access link aggregate since the connection's bottleneck is at this access link, i.e. at the HFS. Thus, working on transport layer connection level does not provide a significant advantage.
So, operating on transport layer connections would come at an increased computational cost and would unnecessarily limit the impact of end host prioritization by CC, while not achieving a significant advantage.

The operation on AN level, i.e. recognizing the traffic type of aggregates destined to an AN, does not make sense. We give two reasons: First, for two-level topologies this level is the BNG downstream interface, so there is nothing to adapt. Second, the CCA do not compete freely within this aggregate, since the connections' bottlenecks are defined by the lowest level in the hierarchical scheduler, which is the access link. Therefore, a single connection cannot dominate the aggregate, neither on regarding bandwidth nor buffer utilization. This means that the traffic differentiation, the implicit signaling RADICCO shall exploit, cannot work on this aggregation level.

In consequence, operating on access link aggregates, i.e. on a per-subscriber basis, is the only reasonable option.

3.8.2 Extent of State Updates

In the RADICCO algorithm, the states and the effective rates are only updated for the leaf node triggering the update and all its ancestor nodes. In most cases, these changes in theory result in rate changes for all other nodes, too. For instance, when a leaf node becomes active, the target rates of its sibling nodes are not updated, although during overload this event results in all siblings' effective rates being reduced. The same applies to its parent: Due to the new leaf, the parent obtains a higher effective rate at cost of the effective rates of all its siblings and their subtrees. Therefore, in an overloaded topology, one update may cause changes to the effective rates of all nodes.
We deliberately designed RADICCO to not immediately reflect these changes. As we will show in the following, the computational effort is substantially reduced, and the drawback in schedule fairness is negligible.

As we will discuss in the evaluation (Section 4.1.2), the theoretical asymptotic worst-case complexity of RADICCO is $O(N)$ (N being the total number of nodes), and it can be expected to be constant, i.e. $O(1)$ in practice. This allows deployment in practice. If the state update involves all affected nodes, the theoretical asymptotic worst-case complexity increases to $O(N^2)$, but also in practice the computational effort would be scaled by N, resulting in $O(N)$. For the typical numbers of nodes and packet rates in today's access networks, this is not feasible for implementation.

Nevertheless, all DRR schedulers only consider a flow's weight when serving it, and their list of flows has no specific order with respect to weights. Thus, our approach of the reduced update set does not make any difference in schedule for DRR-based schedulers. This is different for timestamp-based schedulers, which reorder their priority list after a dequeue operation, so if RADICCO updated the state of the whole scheduler tree, it might result in a difference in schedule.
So, if RADICCO is applied to a DRR scheduler, there is no drawback at all. If RADICCO is applied to a timestamp-based scheduler such as WF^{2}Q+, as we do for the evaluation presented in Chapter 4, the deterioration is no worse than using a DRR scheduler. Moreover, ISPs and equipment manufacturers seem to consider the precision achieved by a DRR scheduler as being sufficient since many devices designed for BNG deployment use DRR schedulers (see Section 2.2.6). As we show in Chapter 4, the evaluation shows no hints that this compromise caused deterioration of fairness.

Summarizing, limiting the state updates of RADICCO to the triggering leaf node and its ancestor nodes substantially reduces the computational effort while the drawback in schedule fairness is negligible.

3.8.3 Filling Up the Rates of Background Traffic

In RADICCO, reduced target rates for background traffic are calculated. There are two strategies possible for handling these target rates in the scheduler:

a. The target rates are strictly enforced, even if in consequence bandwidth remains unused. The final target rates equal the initial target rates.

b. To maximize utilization, the target rates are filled up to the maximum that the current situation allows to transport respecting the allocations for foreground traffic. So, the final rates may be higher than the initial target rates.

When assessing the two strategies, it is important to consider the consequences for the two different types of schedulers that may be extended by RADICCO:

1. Inherently rate-limiting schedulers for which the rate limit and the weight are represented by one variable. Our evaluation is based on a scheduler of this type.
2. Work-conserving schedulers that are deployed in combination with a corresponding hierarchical rate traffic shaper. In this case, the weight is applied to the scheduler and the rate limit to the corresponding rate shaper. DRR-based schedulers are often used in such combination.

For a scheduler of the first type, there is only one way of implementation and it is the same for both strategies: RADICCO calculates and configures the rates, the inherently rate-limiting scheduler enforces the effective rate in terms of both the weight and the rate limit.

In contrast, for a scheduler of the second type with a separate rate shaper the apt implementation differs for the two strategies:
The strict strategy must be implemented by applying the rates to both the scheduler and the rate shaper. The rates in the rate shaper must be adapted to enforce the rates when additional bandwidth is available, while the adaption of the weights in the scheduler is required to enforce appropriate cuts when the available bandwidth is less than the sum of the configured shaper rates. Although this approach would also work for the fill-up strategy, this strategy is best implemented by only adapting the scheduler's configuration of weights while leaving the configuration of rate limits of the rate shaper untouched. With such configuration, the effective rate will fill up otherwise unused bandwidth. This works because in such situation the sum of all target rates is lower than the available bandwidth, the scheduler is work-conserving and only background traffic can arrive at the scheduler with higher rates than the target rate. So, there are points in time when only background packets are available for scheduling, which then are scheduled by the work-conserving scheduler independently of any weights. Therefore, when only the scheduler weights are adapted by RADICCO, background traffic automatically fills up remaining capacity. This approach works for both edge and core nodes.

In the following, we discuss the two options in filling up rates of background traffic separately.

Option 1: Strict Enforcement of Background Rates

We first inspect the approach of accepting underutilization and setting $r_{i,j}^{BG}(t) \leq w_{i,j}^{BG}(t)$. This approach works for all schedulers and the rate decrease for the background leaf nodes is as smooth as the actual adaptation algorithm. Nevertheless, this approach has substantial drawbacks. First, if the HFS is implemented as a combination of a work-conserving scheduler and rate shapers, this approach requires to not only adapt the scheduling weights, but also the maximum rates of the respective rate shapers. Nevertheless, we consider this feasible. Second, it leaves scarce bottleneck capacity unused and thus by design does not achieve our second goal. Moreover, reducing utilization results in extending the duration of the peak load situation, further increasing the impact of this deficiency. Third, the impact of foreground subscribers falsely recognized

as background subscribers is unnecessarily increased, i.e. by applying RADICCO the QoE is unnecessarily deteriorated.

Option 2: Filling Up the Rates of Background Traffic

For RADICCO, we opt to not strictly apply reduced target rates of background traffic. To the contrary, as long as there is bandwidth left, background traffic receives more bandwidth than the reduced target rate, during Underload (according to Section 3.7.7) even the maximum possible rate.

This design results in a smooth transition between *StandardOperationMode* and *Adapting-OperationMode*. So with this design, it does not matter when exactly a parent node enters *PeakLoadState*, so RADICCO is independent of the specifics of the definition and detection of peak load. Thus, RADICCO's dependencies on the system to be extended by RADICCO are restricted.

Filling up background traffic rates supports achieving our second goal, i.e. high bottleneck utilization, without putting any risk on achieving our primary goal of improving the overall QoE for two reasons

- If recognized foreground traffic increases, the fill-up capacity is immediately allocated to it. So, there is no deterioration of traffic recognized as foreground traffic.
- If there is foreground traffic falsely recognized as background traffic, the consequences for the respective subscriber's QoE are mitigated.

This mechanism of instantaneous reallocation of fill-up capacity has an important drawback: Any event of instantaneous removal of fill-up capacity induces a risk of substantially disturbing the control loops of the background traffic CCAs. The CCAs detect a significant increase in congestion and may react undesirably vigorously. Depending on the background traffic's CCAs, rapid decreases of the effective rate due to the removal of fill-up bandwidth may result in throughput inferior to the target bandwidth. Nevertheless, this happens only for true, i.e. correctly recognized, background traffic and therefore does not impact the potential benefits for QoE. Moreover, such overly vigorous reactions are only likely if the change by removing the fill-up capacity is substantial. The average amount of removed fill-up capacity decreases with increasing number of active subscribers at an AN and with the number of background subscribers, so small topologies are more likely to be affected than larger ones. Filling up target rates also has the implementation advantage of only scheduling weights needing to be adapted, also in case maximum rates are enforced by separate rate shapers.
Summarizing, filling up the rates of background traffic where possible increases the bottleneck utilization in most scenarios and if it affects QoE, its effect is positive.

3.8.4 Calculation of Target Rates of Background Leaf Nodes

By the target rate calculation for background leaf nodes we aim to achieve three goals:

- No background connection shall be starved.

- Sensitive low-bitrate services, e.g. standard VoIP or gaming traffic, shall not be impacted by RADICCO.
- To keep utilization high, the reduction is carried out slowly.

Therefore, the rate calculation consists of two components:

1. A minimum background target rate.
2. A function to derive the target rate to be considered when scheduling the next dequeue event.

We will further motivate and discuss both components in the following.

3.8.4.1 The Minimum Target Rate for Background Traffic

We aim to design RADICCO so that no background connections are starved for two reasons: First, we assume that a user or a mechanism at the end host may change the CCA whenever the performance of a flow becomes QoE relevant. This obviously does not work if a connection was disrupted due to receiving no capacity from the scheduler. Second, when a connection is reset due to a timeout, there may be a QoE impact even for background traffic, e.g. because a warning message pops up. Preventing connections from starving can be achieved by introducing a minimum rate.
The second reason for introducing a minimum target rate for background traffic is a deficiency of the used traffic type recognition algorithm (see Section 3.8.5). While both foreground and background traffic can be reliable recognized if the bottleneck is in the scheduler, the rare third type of traffic, i.e. traffic that is rate-limited at another place, cannot be recognized as such. Yet, there are services such as VoIP and video conferencing, that create such traffic. Introducing a minimum rate protects a subset of this group, namely the subscribers consuming sufficiently little bandwidth, from negative impact by RADICCO. Since the target rate for background traffic is not enforced directly, this minimum target rate must be greater than the minimum rate to be achieved multiplied by the expected maximum overall overload. The overall load corresponds to the product of all expected local load factors O_i of the ancestor nodes of the respective leaf node (see Section 3.7.7).

The minimum target rate for background traffic can be configured per leaf, relative to the respective access link's capacity, or globally for all leaf nodes. For our simulations, we choose to configure a global parameter bg^{min}.

The value of bg^{min} or, in case of individual parameterization, the values of $bg^{min}_{i,j}$ of subscribers i, j recognized as background subscribers, impact the amount of capacity RADICCO can reallocate from background to foreground traffic. Therefore, low values may allow RADICCO to improve QoE also for high overload and / or a low fraction of background traffic. It therefore is a configuration option that needs to be adapted to the expected traffic at deployment.

3.8.4.2 The Adaptation Function for Background Target Rates

The adaptation function for background target rates must ensure a smooth transition from a previously assigned foreground rate since any rapid reduction of the assigned rate would often result in undesired substantial load changes or even oscillating behavior.

Such behavior can be caused by background CCs varying their reaction to increased delay based on the extent of the change: Slow changes are compensated by reducing the rate as necessary for fair link sharing with other background flows. In contrast, fast changes are interpreted as starting foreground traffic, so the background CCs yields. This second reaction is for some background CCAs very drastic, resulting in an abrupt reduction of the sending rate to a value close to zero, e.g. the hibernation behavior of uTP (see Section 2.4.3.5).

So, the target rate of a background subscriber should be gradually adapted at every dequeue event during *AdaptingOperationMode*. To not excessively increase the complexity of the overall algorithm, the calculation should require constant computational effort. We shortly discuss the degrees of freedom in designing the adaptation function:

- Begin of adaptation, i.e. definition of zero of the function.
- Input domain of the function, e.g. time, transmitted packets, or transmitted bytes.
- Shape / formula of the function.

To allow smooth transition from the foreground target rate, the rate adaption must start when the node enters *AdaptingOperationMode* and is recognized as background subscriber at the same time. This means that the domain of the target rate adaptation function is specific to that node, i.e. we manage a function

$$w_{i,j}^{BG}(t_{i,j})$$

for every background node in *AdaptingOperationMode*. $t_{i,j}$ is a node-local time domain that starts with the begin of the current phase rate adaption at this node. To ease reading, we will use the simpler notion $w_{i,j}^{BG}(t)$ instead of $w_{i,j}^{BG}(t_{i,j})$.

The domain of the rate adaptation function must be chosen considering that this rate reduction also is a signaling to the sender(s). Therefore, the number of transmitted bytes is not a good choice since this value is not significant for known background CCAs. The decision between time and packets as measure of distance is not obvious since background CCs, on the one hand, are based on measuring delay and its changes in time, but on the other hand the sender CC gets a new measurement only if a packet is served by the scheduler (after some delay). Our calibration simulations showed that the packets domain is the more robust choice and finally allows stable operation with faster decreases than functions using time domain inputs.

Regarding the formula of the function, we achieved good results with a linear decrease. This is not surprising since this corresponds to an approximately constant relative target rate reduction in time, which again corresponds to an about approximately multiplicative reduction factor per RTT for the sender' CCA.

The effect of a reduce factor β_{reduce} of course depends on type and parameterization of the CCA applied by the sender, which cannot be expected to be known and, moreover, may evolve in the

future. Assuming that a background CC tolerates some relative change in available bandwidth during one RTT seems to be reasonable and straight-forward. Moreover, to be successful, any background CCs must allow sharing a link with newly starting as well as already existing background connections. With that respect, today's wide-spread background CCs such as uTP can be expected to serve as guidance for the design of potential new algorithms. Therefore, we feel confident that in particular the type of reduce function but also the order of magnitude of the reduce factor will effectively work for many scenarios and probably even for other, new background CCs.

3.8.5 Traffic Type Recognition

To recognize the dominating behavior of a subscriber's aggregate, the CCs' behavior and the assumed topological properties are exploited. Foreground CCAs fill the buffer at the bottleneck and frequently cause packet losses there. Background CCAs, i.e. LBE CCAs, do not use a significant amount of buffer space at the bottleneck and do not cause packet loss at a bottleneck of constant capacity. Moreover, a foreground connection therefore dominates the behavior of an aggregate regardless of the number of background connections (see Section 2.4.2). For traffic in residential broadband access networks, the BNG's hierarchical scheduler usually represents the bottleneck of incoming traffic (see Section 2.1.1). Therefore, observing the traffic on a scheduler leaf node corresponding to a subscriber's access link suffices to recognize the traffic type of any traffic that is limited by this link.

Nevertheless, there is a third, rare group of transmissions. If the sum rate of all transport layer connections of a subscriber is significantly less than his current effective rate, the access network does not constitute a bottleneck for any of these connections and their CCAs do not matter at all. This mainly concerns

- Short transmissions that cannot reach sufficiently big congestion windows to fill the link's capacity and build up a queue.
- Transmissions that are limited at another place, e.g. at a peering point or by the sender itself.

Technically, distinguishing foreground from background traffic is possible by observing the queue's behavior for some time. This observation shall achieve several, partly contradicting goals:

Low Runtime Effort Since all leaf nodes must be continuously observed, the algorithm must cause little computational effort.

Fast Decision The delay in recognition should be small since the speed of the recognition of a subscriber's traffic type heavily impacts

- The potential bandwidth reallocated by RADICCO from background traffic to foreground traffic.
- Unwanted QoS, and thus QoE impairments for foreground traffic due to continued false recognition as background traffic.

Low False Positives The impact of foreground traffic recognized as background traffic differs depending the traffic and on the duration of the false recognition. In some cases, e.g. for short transmissions, it causes no significant QoS or QoE degradation for the affected connections, but if long lasting connections were wrongly and persistently recognized as background, this would result in severe QoS and QoE deterioration compared to the standard HFS.

Low False Negatives If background traffic is recognized as foreground traffic, there is no disadvantage compared to the standard HFS, but the gain of RADICCO is diminished compared to a theoretical optimal traffic type recognition algorithm.

There are two basic mechanisms that fulfill the low effort criterion, and which we examined for their performance regarding false negatives and false positives: The first candidate just observes if there is a packet drop during a time interval $tt\_recog_{i,j}^{obsDrop}$. The second candidate observes if the buffer usage is above a defined threshold $tt\_recog_{i,j}^{absQThresh}$ during a time interval $tt\_recog_{i,j}^{obsDuration}$. In the following, both approaches are shortly presented and discussed (Sections 3.8.5.1 and 3.8.5.2). Finally, the fundamental drawback of approaches based on buffer observation only is described and analyzed (Section 3.8.5.3).

3.8.5.1 Drop-based Traffic Type Recognition

The lack of packet drops may be considered the most decisive criterion for identifying traffic being background, i.e. not loss-controlled. This is true for a wide range of scenarios: Usually, background CCAs keep the buffer usage very low. Assuming a background connection in congestion avoidance, a packet loss can only happen if the available bandwidth at the bottleneck, i.e. the scheduler, is reduced that heavily that the buffer flows over before the sender's CCA can react. So, usually there are no packet drops at the leaf node's queue unless a new aggressive connection starts, i.e. the traffic type of that flow changes. Moreover, the approach to monitor packet drops requires setting just one parameter per subscriber, $tt\_recog_{i,j}^{obsDrop}$.

The parameter $tt\_recog_{i,j}^{obsDrop}$ also defines the detection delay, so it should be configured as low as possible. However, to reliably avoid false positives, this parameter must be set to a value bigger than the interval between two drops for foreground traffic. Loss distances are expected to be up to 15 seconds for TCP Cubic and scale linearly with the BDP for TCP NewReno (see Section 2.4.3), which is still used by a relevant fraction of Internet servers (see Section 2.4.3.1). For 20 Mbit/s, 100 ms RTT and perfect buffer sizing, TCP NewReno provokes a packet loss only about every 60 seconds. Even if taking the risk to frequently falsely recognize TCP NewReno traffic as background traffic, 15 seconds are a long delay for detecting background. This especially reduces RADICCO's potential in scenarios where the background traffic does not consist of long-lasting connections of few heavy users, e.g. P2P traffic, but short transfers of many different subscribers, e.g. software update downloads.

3.8.5.2 Buffer-Usage-based Traffic Type Recognition

Background traffic can be recognized with low computational cost by checking if the buffer usage exceeded a threshold $tt\_recog_{i,j}^{absQThresh}$ within the last interval $tt\_recog_{i,j}^{obsDuration}$. This approach requires two parameters to be tuned but it can be expected to work well, i.e. to not result in false positives, at a much lower delay. For this approach, the delay is defined by $tt\_recog_{i,j}^{obsDuration}$. Although the monitoring is performed using a specific absolute threshold $tt\_recog_{i,j}^{absQThresh}$ for every leaf node i, it is more useful for configuration to use a relative threshold $tt\_recog^{relQThresh}$ that defines the threshold as fraction of the buffer size. Consequently, the observation interval can be defined globally as $tt\_recog^{obsDuration}$ Current best practice for buffer sizing is configuring the buffers size to $bandwidth * RTT_{estimatedMax}$ (see Section 2.4.4). The factor β, which is used by traditional AIMD CCAs for cwnd reduction on congestion, defines the buffer usage after the cwnd reduction (see Section 2.4.1). So, with correct buffer sizing and for TCP Cubic, the buffer usage after *cwnd* reduction is still $0.4 \cdot BDP$, i.e. corresponding to 40 % of the buffer and 40 % RTT queuing delay. A lower RTT than assumed for buffer-sizing results in an even higher standing queue, e.g. if the RTT is only half as big, the standing queue is 55 % of the buffer, corresponding to 110 % RTT queuing delay. So, for steady state TCP Cubic connections, a $tt\_recog_{i,j}^{relQThresh} = 0.4$ should result in no false negatives regardless of $tt\_recog_{i,j}^{obsDuration}$. Nevertheless, the dynamics in the scheduling system, e.g. leaf nodes becoming active and inactive, result in varying bandwidth allocations.
TCP NewReno uses a β of 0.5, so it exactly empties the queue when the buffer size matches the BDP. Since it moreover increases its cwnd by only one segment per RTT, congestion epochs of TCP NewReno last long (see Section 2.4.3.2). This results on the one hand in probably false recognition shortly after every congestion event but on the other hand in stable foreground recognition during the second part of each congestion epoch. A TCP NewReno connection is recognized as background traffic at $tt\_recog^{obsDuration}$ after its last cwnd decrease if the RTT corresponds to the RTT assumed in buffer sizing. If the effective rate is reduced due to overload, this results in this flow exceeding the recognition threshold $tt\_recog_{i,j}^{absQThresh}$ rather fast. Nevertheless, after the first phase of false background recognition, the buffer usage exceeds $tt\_recog_{i,j}^{absQThresh}$, therefore the subscriber is constantly recognized as foreground subscriber. For a RTT of 100 ms, a rate of 20 Mbit/s, buffer size equivalent to the RTT and a $tt\_recog^{relQThresh} = 0.35$, this period of constant recognition is about 40 seconds.
Although the system behavior is not perfect for TCP NewReno, we accept this because TCP NewReno is of limited importance for high volume transfers due to its slow reaction to changed conditions.

In our calibration simulations, we found

$$tt\_recog^{obsDuration} = 2s \quad \text{and} \quad tt\_recog^{relQThresh} = 0.35$$

necessary to reliably recognize foreground traffic as such.
So, the buffer-usage-based traffic type recognition also provides low runtime effort and a much faster decision than the drop-based traffic type recognition. For traffic having its bottleneck at the scheduler, it falsely recognizes foreground traffic as background traffic almost only due to preceding idle times (due to our choice of traffic type recognition, see following Section 3.8.6) but not during ongoing transmissions. The number of false negatives, i.e. recognizing background traffic as foreground traffic is higher than for the drop-based approach, especially for TCP Vegas

traffic, but we accept this as a minor disadvantage.
In case an AQM is configured for the leaf queues, the threshold $tt\_recog^{relQThresh}$ must be adapted to the operation of the AQM.

Static Buffer Usage Threshold

Although the target rate of background leaf nodes is reduced, we do not adapt the recognition threshold $tt\_recog^{relQThresh}$ correspondingly. This is a design decision more important than it seems at first sight. If the effective rate decreases with the target rate, the sender's CCA can only react to the change carried out one RTT ago, so the buffer usage is higher than in steady state. So, in that phase, a stable, not immediately reduced recognition threshold helps preventing false negatives, i.e. recognizing background traffic as foreground traffic.
Yet, we also do not adapt $tt\_recog^{relQThresh}$ to the target rate after the target rate reduction phase has been completed because target rates are filled up to utilize available capacity. So, effective rates of background leaf nodes are often substantially higher than the target rates. In that case, background CCAs aiming at keeping the queuing delay constant (rather than the queue size), i.e. uTP, use more buffer space. Moreover, in contrast to effective rates of foreground leaf nodes, the effective rates of background leaf nodes vary more heavily, so, depending on the traffic dynamics, frequently packets temporarily queue up at background leaf nodes.
So, also in the phase of $w_{i,j}^{BG} = bg_{i,j}^{min}$, we avoid false foreground recognition of background traffic by not decreasing the recognition threshold.
Moreover, adapting the threshold to the reduced target rate would not solve the problem of all approaches exclusively based on buffer observation (see next section).

3.8.5.3 Deficiency of Approaches exclusively based on Buffer Observation

The fundamental deficiency of traffic type recognition exclusively based on buffer observation is that traffic that is otherwise rate-limited cannot be detected as such and therefore is not handled appropriately.

Besides the majority of traffic that has its bottleneck in the access network, there also is traffic that either is not limited by a network bottleneck but by the source, or is limited by bottlenecks at other locations in the network, e.g. peering points, the source aggregation network, or within the subscriber's network, e.g. by a slow WiFi link. This type of traffic is rare, but exists, so we detail the consequences of RADICCO for such traffic.

A problem in terms of QoE deterioration arises for traffic meeting the following criteria:

- The rate limit is permanently lower than the subscriber's current effective foreground rate. If it would not, the BNG's scheduler were the bottleneck and the traffic were correctly recognized.
- The rate limit is (frequently) higher than the currently effective minimum background rate. If it would not, the traffic received the rate it requires.
- Traffic is foreground traffic. If it would not, the process described below would apply but would not result in QoE deterioration.

One example for such traffic is true Constant Bit Rate (CBR) traffic with a rate between the subscriber's effective foreground rate and the subscriber's effective minimum background rate. As described in Section 2.3.3, CBR traffic is rare in today's Internet. Moreover, even with RADICCO in the proposed configuration any subscriber will get a rate of $\frac{bg^{min}}{overloadFactor}$. This mechanism can be used to protect low bandwidth CBR traffic such as VoIP.

The effects of RADICCO on traffic meeting the criteria described above are summarized in the abstract Figure 3.6, depicting internal states and bandwidths for a node i, j receiving traffic that is rate-limited to a constant maximum rate r. For simplicity, the figure assumes the effective rate to correspond to the target rate at all times.

Otherwise rate-limited traffic does not build up a queue for effective rates higher than their external rate limit, so by all approaches based on buffer observation, e.g. the ones we presented above, such traffic will be recognized as background traffic after some time, indicated as t_0 in Figure 3.6. While the assigned effective rate is higher than the external rate limit, so until point t_1 in Figure 3.6, the respective leaf node is not continuously active, i.e. its queue will be empty again and again.

Nevertheless, if the lowest effective rate allocated to background leaf nodes is lower than the external rate limit at some time, a queue starts building up once the effective rate decreases below the external rate limit, i.e. after point t_1 in Figure 3.6. In detail, the queue size grows faster than linear between t_1 and t_2 due to the fixed incoming rate and the decreasing effective rate. In this phase, the queuing delay grows even faster than the queue size since both, the queue size grows and additionally the dequeue rate decreases.

After some time, the minimum target rate $bg_{i,j}^{min}$ may be reached if the queue does not build up too fast. This is shown as point t_2 in Figure 3.6. In the following phase, the queue is served at the minimum target rate $bg_{i,j}^{min}$ and the incoming rate corresponds to the rate limit of the traffic, so both queue size and queuing delay increase linearly.

At some time, the queue size exceeds $tt\_recog_{i,j}^{absQThresh}$, shown as t_3 in Figure 3.6. Then, the leaf node is again recognized as foreground leaf node, thus the effective rate returns to $C_{i,j}$. Until this happens, the low effective rate and the, in relation to the effective rate, high buffer usage threshold $tt\_recog_{i,j}^{absQThresh}$ result in high waiting times, i.e. queuing delay. For a leaf node i, j, the resulting maximum delay τ_{max} depends on the relation of the recognition buffer usage threshold $tt\_recog_{i,j}^{absQThresh}$ to its minimum background target rate $bg_{i,j}^{min}$, and the overall load factor, i.e. the product of O_i and O_R in case of three-level hierarchy. The formula is given in Equation 3.36.

$$\tau_{max} = \frac{tt\_recog_{i,j}^{absQThresh}}{bg_{i,j}^{min} \cdot \frac{1}{O_i \cdot O_R}} \tag{3.36}$$

$$= \frac{C_{i,j}}{bg_{i,j}^{min}} \cdot RTT_{buffEst} \cdot tt\_recog_{i,j}^{relQThresh} \cdot O_i \cdot O_R \tag{3.37}$$

If the threshold is defined relatively, the access link's capacity $C_{i,j}$ and the RTT estimation applied at buffer sizing, $RTT_{buffEst}$ are needed, see Equation 3.37. The presented formula is a simplification for an theoretical steady state. In real systems, many parameters vary over time, in particular $o_i(t)$ and $O_i(t)$, so also τ_{max} is a function in time. Nevertheless, this formula allows to estimate the maximum delay to be expected for a scenario.

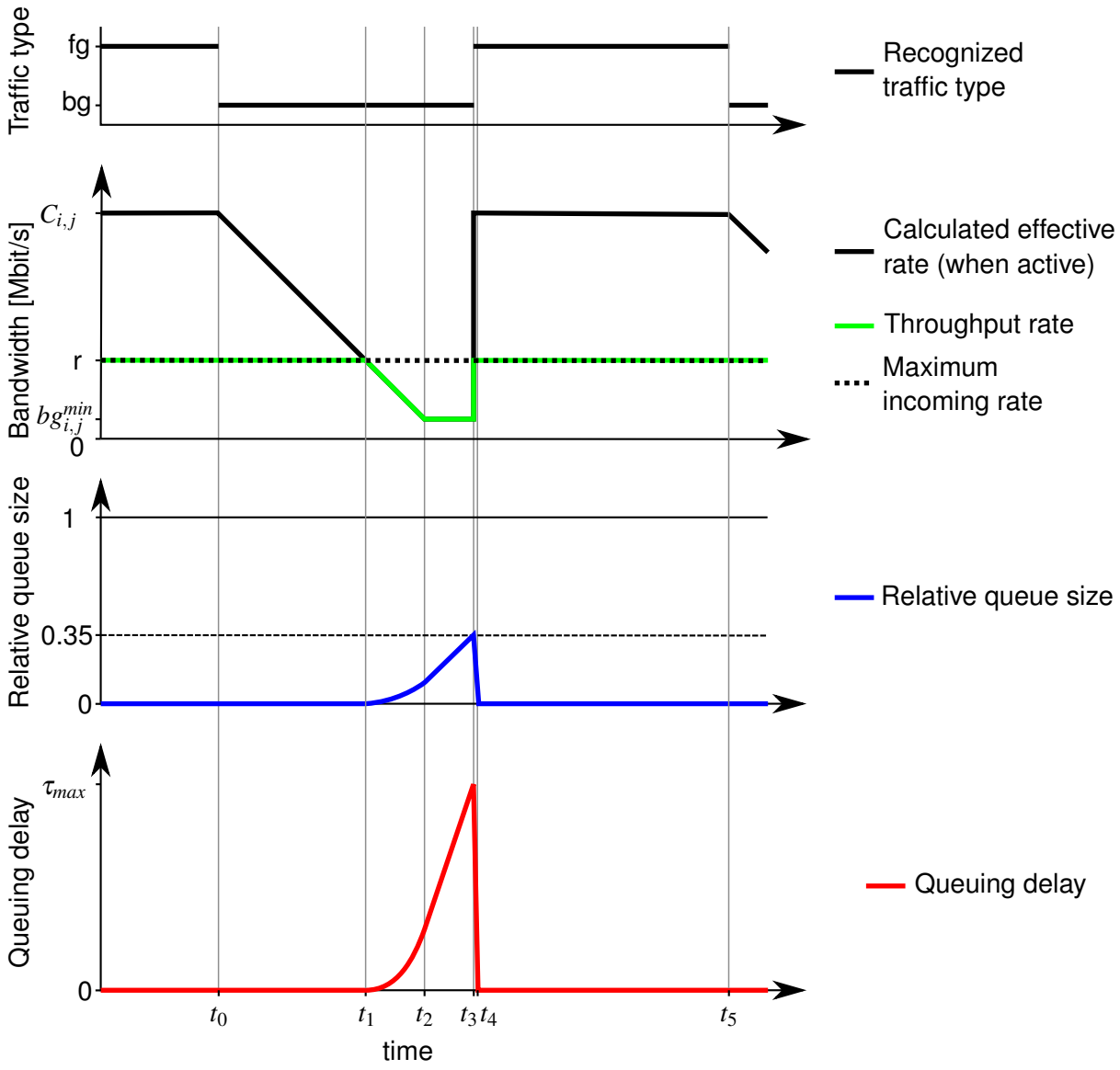


Figure 3.6: Schematic of the evolution of states of a leaf node receiving otherwise rate-limited traffic only

After the node is recognized as foreground leaf node, the queue drains fast and linearly, thus reducing queuing delay at the same speed. At some point in time, t_5 in Figure 3.6, the queue is emptied again. Since, again, the assigned effective rate is higher than the maximum incoming rate, node i, j is not continuously active any more. So, after some time the node is again recognized as background leaf node, depicted as point t_5 in Figure 3.6, and the process started at t_0 repeats.

So, traffic that is rate-limited at another location is not recognized by RADICCO traffic type recognition as a third traffic type. For a certain range of incoming rates, recognized traffic type oscillates between background and foreground. While the average queuing delay may be acceptable, the oscillation results in huge queuing delay for a short period within each oscillation cycle.

This issue can only be solved by a more sophisticated traffic type recognition algorithm that captures and considers the consumed rate. Nevertheless, we consider the capability to detect rate-limited traffic desirable for any deployment. To that respect, several approaches can be examined. First, it might be sufficient to detect phases of inactivity, i.e. when the queue is empty, during rate reduction. If the traffic were limited by the access network's scheduler, the queue would not run empty when the effective rate is reduced. Second, maybe existing counters in products for BNG deployment can be used to determine the actual throughput during foreground phases. If the throughput is significantly lower than the effective rate during that interval, the subscriber's traffic is also not limited by this scheduler. Both approaches generally suffice to reliably detect otherwise rate-limited traffic.

In this thesis, we will evaluate prevalence and extent of this issue, but we consider developing such a sophisticated traffic type recognition algorithm for this rare traffic type out of the scope of this thesis.

3.8.6 Initial Traffic Type

For the design of RADICCO, we must define when a subscriber is recognized as idle and a policy for the initial traffic type when a subscriber becomes active again. While a general, globally accurate technical definition of idle or active is hard or impossible to give, from a scheduler's perspective the only obvious, simple and cheap (in terms of effort) solution is to define idle by link inactivity exceeding a time threshold.

When a subscriber becomes active after an idle phase according to that definition, there are four basic options for the initial traffic type:

- Foreground.
- Background.
- Unchanged, i.e. the same as during the last active phase.
- Ignore, i.e. do not handle idle time specially.

While the best choice mainly depends on the transported traffic, some mechanism-dependent advantages and disadvantages exist, which we shortly discuss.

Always starting with traffic type foreground will, on the one hand, waste a lot of bandwidth for every starting background transmission because it would receive a foreground fair share for $tt\_recog^{obsDuration}$ and, after that, would receive for another non-negligible time still more bandwidth than the background minimum $bg_{i,j}^{min}$. So, overall there is significantly less bandwidth available to RADICCO to be shifted from background to foreground traffic than could be expected. On the other hand, this approach makes sure to not disadvantage starting foreground traffic, especially single short transmissions.

Always starting with traffic type background poses the question of the initial target rate. If starting with maximum target rate, short foreground transmissions will receive maximum service but background transmission will be allowed to occupy a rather large fraction of bandwidth at the beginning. If starting with the minimum background target rate $bg_{i,j}^{min}$, starting foreground transmissions will have to fill the buffer over $tt\_recog_{i,j}^{absQThresh}$, before they are detected as foreground traffic. This is achieved rather fast in slow start, limiting the disadvantage. Often, the effective rate is higher than the target rate, speeding up the process. Still, for typical buffer sizes and our proposed parameters, this may take several RTTs.

The third option is to leave the subscriber traffic type untouched after phases of inactivity. Generally, this option is preferable if there is a high probability that the new activity is of the same traffic type as the last one. Today, this if often the case because most traffic is caused by human activity and humans tend to pursue activities rather for hours than for seconds. For example, for DASH video streaming, each burst should and probably will be recognized as foreground traffic and a session typically comprises thousands of bursts.
Again, for the background case there is the choice to keep the assigned target rate or to reset it to the maximum target rate. Since subsequent objects may be transmitted using an existing transport layer connection with an already adapted congestion window, it may make sense to reuse the formerly assigned target rate. In contrast, resetting the target rate to the maximum after each idle period would again unnecessarily restrict RADICCO's rate adaptation scope, assuming that the probability that the traffic type does not change is high.
With a significant number of low rate but persistent connections (see Section 3.7.4.1), this method tends to recognize traffic as background traffic.

The option of not handling idle times special was discussed in Section 3.7.4.1.

Summarizing, the choice of the initial traffic type must depend on the probability of traffic type changes, i.e. on the transported traffic. Therefore, the choice itself cannot be verified by simulation without knowing (or estimating) the probability of a subscriber changing his traffic type.

4 Evaluation

In this chapter, we will evaluate RADICCO as a solution to the problem stated in Section 3.2. In the following, we will first analyze the RADICCO algorithm to show that it meets the qualitative objectives identified in Section 3.4.1. The remainder of this chapter is dedicated to the evaluation of the quantifiable performance of RADICCO regarding the objectives defines in Section 3.4.2. We present the approach used in simulative evaluation, the performance metrics, which are aligned to the objectives, and then examine RADICCO's behavior regarding four selected traffic types.

4.1 Evaluation of Qualitative Objectives

4.1.1 Network Neutrality

It is important to emphasize that RADICCO does not impact network neutrality. We want to highlight the main arguments here.

- RADICCO only reacts to the senders' obvious, i.e. easily detectable behavior. RADICCO interprets the lack of aggressive rate increase as identifying background traffic. This is no misinterpretation as we argued in Section 3.6.3.

- RADICCO does neither cause data discrimination nor service discrimination, it is only based on the sender's behavior. Any service, any data can be transmitted as background or foreground, this decision is up to the sender. This is a crucial distinction compared to DPI-based BE priority differentiation as partly deployed today.

- The bandwidth reduction for background traffic is not enforced if a sender does not cooperate as expected. In more detail, if a sender does not react to the reduced service rate by swiftly reducing its sending rate, the sender will experience no loss and will be recognized as foreground subscriber and accordingly served with full rate.
  Assume a subscriber receiving traffic at low buffer usage, so RADICCO will recognize it as background subscriber and when in overload, start reducing the assigned bandwidth. This results in some queue building up for this customer. If the sender does not react to the resulting queuing delay and continues to send with the same data rate, the subscriber will be recognized as foreground subscriber without even one packet being dropped.

- All control is up to the sender, and by that indirectly to the subscriber in many cases. A sender may change its behavior at any time and RADICCO will adapt immediately. Consider a sender applying TCP with a background CCA. If it intends to increase priority, e.g. because it got informed by the receiver that the user brought the corresponding application to foreground, it may change this connection's TCP behavior. For wide-spread OSs such as Linux, this is easily possible.

4.1.2 Sufficient Efficiency

To estimate if RADICCO is sufficiently effective, we derive its computational complexity and compare it with the complexity that we assume being acceptable, i.e. $O(logN)$ or better (see Section 3.6.2).

To estimate the worst-case computational complexity, we analyze the effort necessary for each adaptation event and the worst-case frequency of adaptation events. We provide a theoretical analysis on the resulting additional worst-case complexity per dequeue-event, as well as a practical analysis taking into account external limitations.

Due to RADICCO's design shown in Algorithm 3.3, each rate adaptation event, i.e. each call of the `updateAdaptation()` function, results in L calls to the `updateState()` function and L calls to the `updateEffectiveRate()` function with L being the number of hierarchy levels of the scheduler. Each of the state update calculations can be executed using a constant number of calculations if state changes are propagated. The calculation of the effective rate of one node also requires only a constant number of calculations.
Therefore, the computational effort caused by an adaptation event does not depend on the number of nodes in the hierarchical scheduler but on the number of hierarchy levels only. So, the worst-case complexity of one adaptation event is $O(L)$.

Adaptation events are triggered by dequeuing a leaf node, i.e. frequency $O(1)$, but also by a packet arrival at an inactive leaf node, i.e. whose queue is empty at that time. Theoretically, all $O(N)$ leaf nodes can become active between dequeuing two packets from the root scheduler, resulting in an overall worst-case additional complexity per dequeue event of $O(L \cdot N)$.
This theoretical complexity is not feasible for implementation in real equipment.

Nevertheless, this is not the complexity relevant for implementation and deployment. For implementation, three aspects substantially limit the number of potential rate adaption events between two dequeue events:

1. Rate control of transport layer CC.
2. Self-clocking of transport layer CC.
3. Rate-limited ingress interfaces.

First, the CCAs prevalent in today's Internet increase packet rates carefully, so on average the incoming bandwidth is not significantly higher than the bandwidth of the outgoing downstream interface. Second, the deviation of the incoming packet rate from its average is small since the packets of every transport layer connection arrive about evenly spaced due to the self-clocking mechanism of prevalent CCAs, e.g. standard TCP. Self-clocking works very well

at the BNG because there is no shared queue, which would result in burst arrivals, but the hierarchical scheduler that schedules each access link capacity separately. Third, any network device deployed as BNG is attached to the provider core or aggregation network with limited bandwidth only. Moreover, the single line card operating the respective downstream interface and executing the packet scheduling is attached to the device's backplane with a specific bandwidth, too. At least the line card's backplane interface typically is not much faster than the outgoing interface speed, i.e. there is a small constant that describes the relation between ingress and egress capacity. Therefore, in practice the number of rate adaptation events between two dequeue events is limited by a small constant, i.e. it is $O(1)$.
So, the practical worst-case complexity per dequeue-event is $O(L)$, with L typically being two or three. This complexity is expected to be feasible for implementation, maybe even for updating existing and deployed devices.

Summarizing, while in theory the worst-case complexity of RADICCO is $O(N \cdot L)$, which prohibits implementation in real equipment, the practical worst-case complexity is $O(L)$, which allows for implementation. Since the number of hierarchy levels in hierarchical access networks is small, typically smaller than four, it could even be argued that RADICCO's complexity is constant for schedulers at BNGs.

4.1.3 Smooth Rate Allocations for Foreground Traffic

Since RADICCO changes the rates during operation, rate allocations for foreground traffic defined by RADICCO may include faster or more abrupt changes than the rates allocated by a standard HFS. The events that could potentially cause differences in smoothness compared to a HFS's schedule are listed below:

1. An additional subscriber becomes active.
2. Another subscriber's rate is reduced due to a background leaf node's target rate reduction.
3. Another subscriber's recognized traffic type changes.

We will discuss these three cases in the following.

With a HFS, when a subscriber becomes active during overload, the effective rates of its sibling nodes are reduced.
With RADICCO, this is the same if all respective subscribers are recognized as foreground subscribers. If some of them are recognized as background subscribers and their target rates are smaller than their capacity, RADICCO behavior depends on the parent node's level of overload according to RADICCO's internal load levels. Without loss of generality, we assume that the parent node is not in *AdaptingOperationMode*, so its current effective rate equals its nominal capacity. If the parent node is in Underload, there is no change compared to the HFS. If the parent node is under Light Overload, the effective rates of background sibling nodes is heavier reduced than the effective rates of foreground siblings. In consequence, the change for the foreground siblings is smaller than if applying a HFS. The effective rates of all foreground subscribers may even remain at their respective access links' capacities, the maximum possible rates. If the parent node is under Heavy Overload, the effective rates of all its child nodes are reduced. Since the share of foreground traffic is higher than it would be with a HFS's fair allocation, also the proportional fair reduction of a foreground sibling node is higher than by a HFS. Nevertheless,

for RADICCO, the resulting effective rate of a foreground sibling node is still higher than its effective rate allocated by a HFS. So, in most cases a subscriber becoming active results in smaller changes in rate allocations of foreground traffic than caused by a HFS. If the change is larger than the change of a HFS, still a higher rate is allocated, which probably results in an improved QoE, the primary objective of RADICCO.

Reducing the target rates of background subscribers is a feature of RADICCO only, so a comparison with HFS is not possible. The reduction of the target rate of a background leaf node is carried out in small steps, but these small steps become irrelevant at load level Underload and have less importance at load level Light Overload. Thus, the target rates of background subscribers are of importance to foreground rate allocations only at load level Heavy Overload. So, if foreground rate allocations are changed due to a target rate reduction of background, they first, currently receive more bandwidth than with HFS so it always is a beneficial change. Second, RADICCO target rate reduction for background leaf nodes is carried out in small steps, thus the rate allocation is smooth.

If RADICCO changes the effective rate of a foreground leaf node because it detects a traffic type change for another node, this type change must be from background to foreground. In that case, the effective rate of the considered node is reduced, but this is necessary to assign the appropriate rate to the new foreground leaf node to achieve appropriate QoE. Moreover, the effective rates of all foreground leaf nodes are not lower than if rates were allocated by a HFS. The change from foreground to background has no immediate effect on the node's target rate, so no sibling nodes are affected.
So, a traffic type change from background to foreground results in abrupt changes in effective rates of foreground leaf nodes, but these are necessary to rapidly provide the new foreground node with its due bandwidth. In that case, the secondary objective cannot be achieved because the primary objective of improving QoE has conflicting requirements.

Concluding, RADICCO in most cases achieves this objective. In some cases achieving the primary objective of QoE improvement requires violating this objective.

4.2 Performance Evaluation Approach

The performance evaluation was accomplished by event-driven simulations. In the simulations, we use a HFS implementation (WF^2Q+ extended by an integrated rate-limiting) implemented by ourselves and this algorithm extended by RADICCO. The simulations covered a targeted set of four traffic models, two distinct topologies and two background CCAs. We use a wide range of overload configurations in terms of offered elastic load and share of background traffic since the offered load defines the necessity for bandwidth reallocation and the share of background traffic defines the RADICCO's room for maneuver.

Since RADICCO's approach of exploiting an understanding on transport layer traffic differentiation in packet scheduling is a novel approach, there is no reference mechanism aiming for the same objectives. This evaluation measures the changes in performance that RADICCO causes compared to a HFS to prove the benefit of RADICCO.

In RADICCO, the traffic type recognition and the rate adaptation interact indirectly via the senders' reactions to the rate adaption. Therefore, its design is a trade-off between stable and reliable traffic type recognition and speed and extent of rate reduction of background traffic. So, any evaluation must consider the system as a whole, e.g. it makes no sense to assume perfect recognition to estimate the maximum benefit that can be achieved by a specific rate adaptation algorithm.

4.2.1 Simulation Utilizing Wide-spread Congestion Control Implementations

One core concept of RADICCO is to consider the subscribers' traffic behavior for allocating scarce aggregation link bandwidths. So, the performance of RADICCO heavily depends on the actual behavior of the traffic sources, i.e. their CCA. Therefore, for performance evaluation of RADICCO, any modified or abstracted implementation of a CC bears a risk to impair the quality and transferability of the simulation results. We use unmodified wide-spread CC implementations as "black box" modules in our simulations. In consequence, we model traffic on application level, i.e. these models define the points in time at which demands arrive and the object sizes, i.e. the number of bytes to be transferred.

For this purpose, we use the IKR simulation library (IKR SimLib) [158] with the VMSimInt enhancement to integrate virtual machines (available at [159]) running unmodified OSs, which we, IKR colleagues and the author, first presented in [160]. This simulation framework has been used for various research, e.g. our works on CPQs [155] and DCTCP [154] made use of it. This framework provides a precise packet-level simulation: It uses unmodified Linux kernels to provide black box CC implementations translating transmission requests on application level to packets of bidirectional connections traveling through the simulated network topology. This precise packet-based approach means that the simulation effort, i.e. the run time needed to complete a simulation, scales with the number of packets transmitted. This number depends mostly on two factors: the simulated time and the average load. Since for the presented simulations all packets pass the simulated BNG interface and this interface is fully loaded most of the time, run time scales roughly with the BNG interface's capacity. We do not use a fixed BNG interface capacity but capacities calculated according to the targeted overload as described in Section 4.2.3.

The VMSimInt simulation framework supported only TCP sockets as provided by the OS within the virtual machine. For our research on CC-based prioritization we extended this framework by support for uTP since it is the most wide-spread background CCA (see Section 2.4.3.1). uTP is based on pseudo-sockets, i.e. a socket-like interface provided by the libutp library [138] instead of the OS's kernel. Our extension allows uTP pseudo-sockets to be managed and accessed, most importantly written to and read from, within the simulation.

By that approach we use existing, wide-spread CC implementations without any modification from both user and kernel space as black box algorithms for packet generation. We use three unmodified CCs in the simulative evaluation:

- TCP Cubic from Linux Kernel release 4.4.6 [161]
- BitTorrent's uTP provided by libUTP as of Feb 27, 2015 [138]

- TCP Vegas from Linux Kernel release 4.4.6 [161]

We chose Linux kernel 4.4.6 since it was the most recent long term support kernel at the begin of evaluation. It will be maintained by the kernel developers at least until early 2018.

In the following sections, we argue why we chose these algorithms and excluded others.

CCs for Foreground Traffic

TCP Cubic is the default congestion control in Linux. As presented in Section 2.4.3.3, TCP Cubic accounts for a large share of Internet traffic. Moreover, this share mostly origins from Linux servers, so the Linux kernel implementation is by far the most used foreground CC and therefore the most interesting candidate for foreground CC in our simulations.

Traditional AIMD or TCP NewReno is also used by many Internet hosts, see Section 2.4.3.2. Nevertheless, TCP NewReno is not suitable for faster links, or more precisely larger BDPs, since it allocates new bandwidth in congestion avoidance phase unacceptably slowly, again see Section 2.4.3.2. Therefore, we do not expect TCP NewReno to be responsible for large fractions in terms of traffic and do not consider TCP NewReno in our simulations.

The TCP Compound implementation of the Windows OS would also be an interesting candidate since Microsoft Windows hosts use it by default and it is reported to be used by a significant number of Internet hosts (see Section 2.4.3.6). Unfortunately, up to now there is no simulation framework available that allows integrating TCP Compound as implemented in the Windows OS in simulations. Our VMSimInt framework does not support integrating Microsoft Windows instances into the simulation. There exists a Linux implementation of the respective draft [142], but this implementation may vary arbitrarily from the implementation used in the Windows OS. Moreover, Therefore, we refrain from using Compound TCP in our simulations.

Summarizing, we use TCP Cubic as the only foreground CC in our simulations, as it represents by far most of today's foreground traffic volume and maybe even a higher share of traffic of the near future.

CC for Background Traffic

Regarding background traffic, we supervised a Bachelor thesis [139] that dived into the performance of available CCA implementations, making use of our extended VMSimInt framework as described above. In [139], nine CCs were evaluated by their respective available implementations:

- uTP [86]]
- LEDBAT [RFC6817]
- TCP Vegas [162]
- TCP-CDG [163]

- TCP-LP [111, 164]
- TCP Nice [113]
- TCP Veno [165]
- TCP Westwood [166]
- TCP YeAH [167]

Amongst these, BitTorrent's uTP is the best known and likely most wide-spread background CC algorithm. Moreover, in our examination, uTP proved to be the best suitable background CC implementations regarding all applicable metrics, such as speed of yielding or bandwidth utilization when competing with foreground traffic. Note that we found uTP allocating new bandwidth slower and more carefully than other investigated CCs. We do not consider such behavior a drawback for a background CC if the priority goals of fast yielding and low remaining bandwidth consumption are met. This is the case for uTP. Nevertheless, this statement mainly holds for an appropriately low target delay value, see Section 2.4.3.5 for details. We successfully used target values of 10 ms, 5 ms, 3 ms and even 500 μs in our simulations. Generally, we observed that the lower the target value is chosen, the more sensitive the CC behaves. The sensitivity of the system behavior for a given target value depends on the path's jitter and on the capacity of the bottleneck link: The faster the bottleneck, the smaller the delay a packet of a certain size induces. While we used 5 ms in [139], in this work we use 3 ms since we found this value resulting in a robust behavior and low delay, important e.g. for concurrent VoIP traffic, but in a fast detection of foreground traffic by uTP connections.
This uTP implementation implements the zero-cwnd feature we call hibernation (see Section 2.4.3.5), which will affect the system behavior in our simulations.

Among the evaluated CCAs, the second candidate with good background behavior is TCP Vegas, which is a very old and well investigated CC (see Section 2.4.3.4). Other CCs either did not yield as fast or as far as these two candidates. In the evaluation of RADICCO we use a newer Linux kernel than in [139] (4.4.6 instead of 4.2rc8) but there were no substantial changes in the respective kernel code so we do not expect any significant change in the behavior of TCP Vegas.

We therefore decided to not only use uTP with a target value of 3 ms but also TCP Vegas as background CCs in our simulative evaluation of RADICCO.

4.2.2 Simulation Topologies

Although in today's regional access networks there is a broad variety of technologies deployed, see Section 2.1.2, all topologies share some characteristics. They all base on hierarchies and in fact also on oversubscription. One remaining distinguishing feature is the number of hierarchy levels. There are networks that consist only of two levels, i.e. the BNG's outgoing downstream interface directly connects an AN that itself serves several subscribers. But there also are other networks that implement the typical three-level hierarchy.

We therefore model both types of regional access networks:

- Flat, two level topologies, see also Figure 4.1.

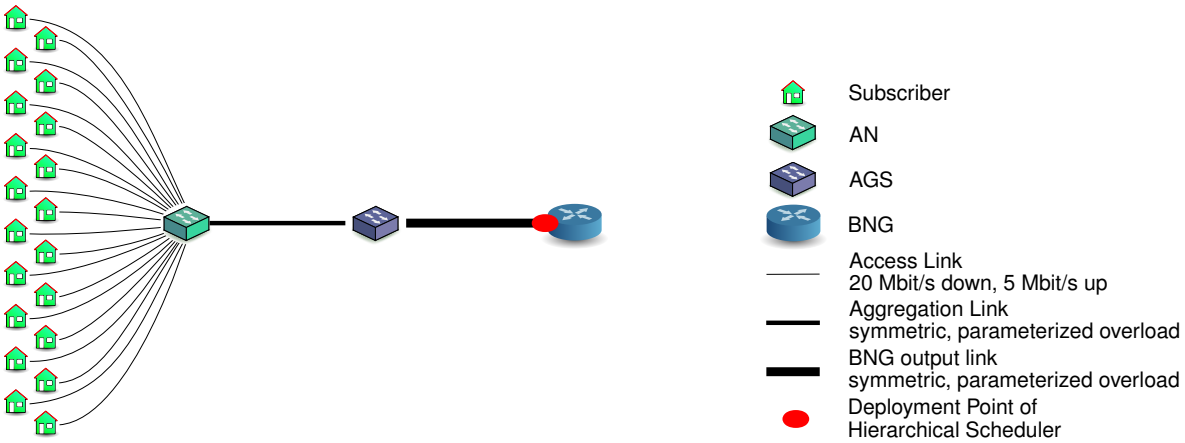


Figure 4.1: Access network topology BROAD used in simulations

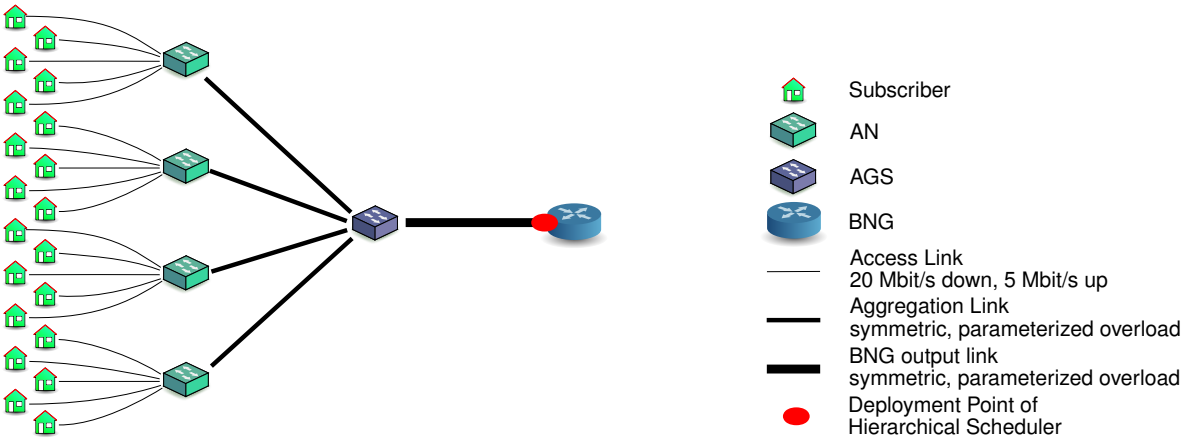


Figure 4.2: Access network topology DEEP used in simulations

- Deeper, three level topologies, see also Figure 4.2.

Technically, all simulations use a common configurable simulation model using a parameterized topology as depicted in Figure 4.3 which allows to model both types of topologies. This model represents the hierarchical network topology fed by one BNG downstream interface, depicted on the left side, and a minimalistic sender side topology, on the right. The upstream topology corresponds to the downstream topology except two aspects:

- There is a simple FIFO queue at the BNG interface output interface.
- The access link is asymmetric and has lower upstream capacity than downstream.

The choice of parameters such as access link capacities is discussed in Section 4.2.5.

With these two types of topologies we evaluate the effectiveness of RADICCO shifting resources from background to foreground on two different levels: For flat access networks, RADICCO is required to operate at the AN scheduler level, i.e. to work on flows corresponding to subscribers, each subscriber being recognized as either background or foreground at a time. So, RADICCO may redistribute resources among sibling subscribers, i.e. from background subscribers to

foreground subscribers attached to the same AN. For deep access network topologies with unbalanced distribution of foreground and background subscribers among ANs, RADICCO is required to operate at the BNG interface level, i.e. to work on flows corresponding to links to ANs carrying mixed aggregates. So, RADICCO may redistribute resources among ANs, i.e. from ANs with a lower foreground share to ANs with a higher foreground share. Moreover, also in this topology RADICCO shall operate on AN level and assign higher rates to foreground subscribers than to background subscribers within ANs serving both background and foreground subscribers.

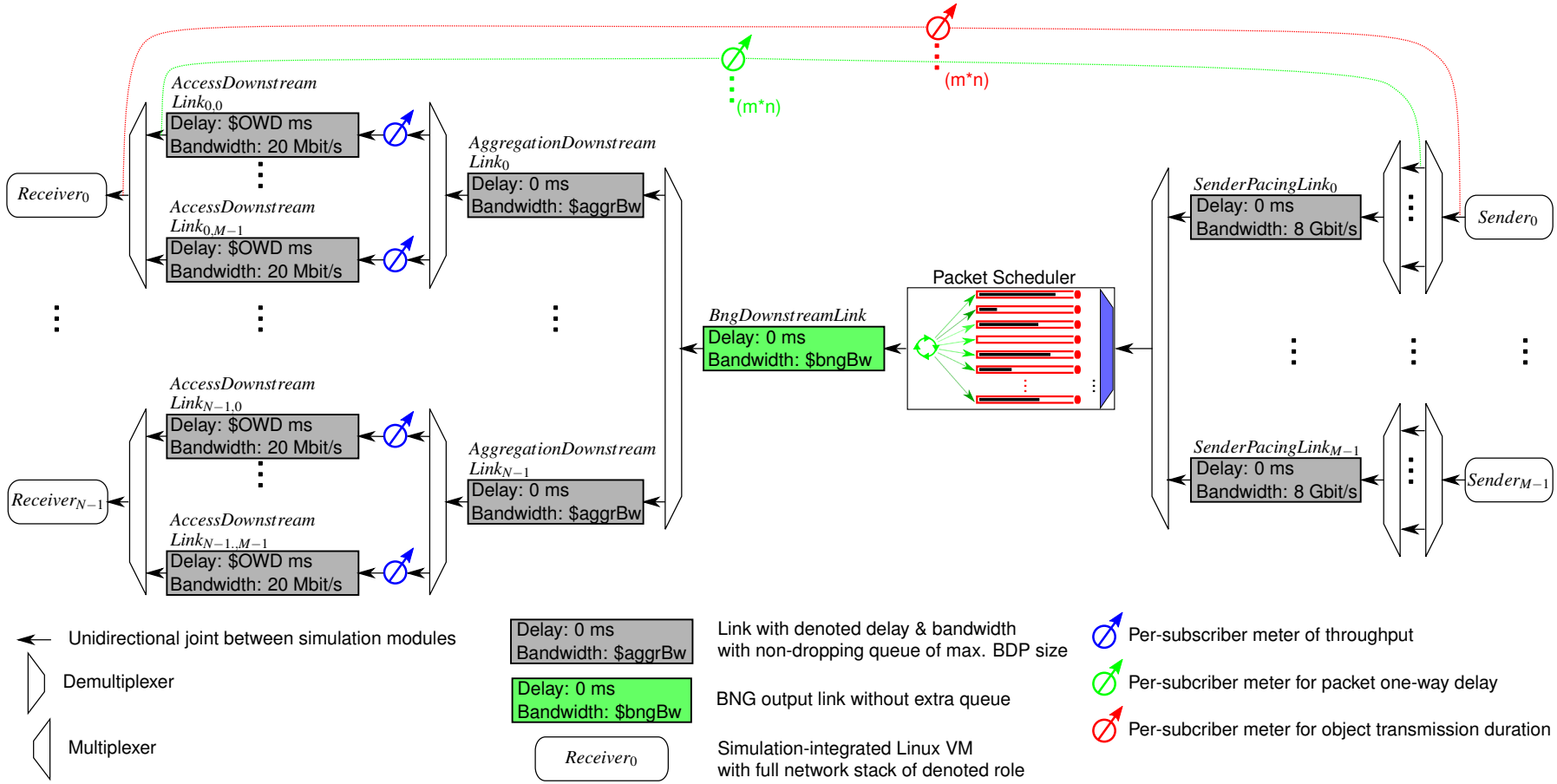


Figure 4.3: Simulation topology (only downstream depicted)

This model only allows for symmetric topologies, i.e. every AN serves the same number of subscribers. While the model allows arbitrary numbers of ANs and subscribers, we only model three instances for the presented simulations: A *BROAD* access network topology with all subscribers served by one AN and either twenty, shown in Figure 4.1, or one hundred subscribers. A *DEEP* access network topology shown in Figure 4.2 with a total of twenty subscribers served by four ANs, each serving five access links.

In most cases, we configure both topologies to contain twenty subscribers. There is one exception which is the web scenario, in which we use one hundred subscribers due to the low average load created by the web browsing traffic model. Note that we do not assume ANs to only serve five to twenty subscribers. But, first, we only need to model active subscribers which in general are far less than the connected subscribers. Second, these topologies are big enough to examine the defined metrics, but also allow to run many simulations, i.e. to evaluate many parameter sets. Third, by using rather few subscribers, we evaluate a lower boundary of performance: A control mechanism as RADICCO functions the better the more fine-grained its control options are. So, by choosing such rather small numbers of subscribers we model a case worse than typical deployment scenarios.

We use symmetric, load-adapted capacities at aggregation links, i.e. for BNG and AGS interfaces since such interfaces typically use standardized fiber-optical technologies that provide symmetric speeds, e.g. Ethernet. Regarding the access links, we decided to configure all subscribers with the same static access link capacity and constant delay.

The uniform capacity differs from real networks where usually a range of capacities is used, be it for technical reasons as for DSL access links or for business reasons as e.g. for HFC or optical access networks (see Section 2.1.2). Nevertheless, for the evaluation of RADICCO there is no significant difference between an AN serving N subscriber links of different capacities or N links all with the average capacity. More importantly, in our model every AN serves the same total subscriber capacity. We assume this also being the goal of real network architects to equally utilize the aggregation links that use standard technology. Therefore, we expect actual topologies to only slightly vary from that ideal of equal sum capacities at each aggregation level.

The static bandwidth is another simplification since the capacity available to BE traffic is the remainder bandwidth after higher priority services, i.e. the ISP's VoIP and VoD services, received their share (see also Section 2.2.7). Nevertheless, we chose to focus on static BE bandwidth in this study since the impact of the CBR VoIP is negligible and the prevalence of ISP-offered VoD services is low compared to other VoD providers such as Netflix, YouTube or Amazon Video. Moreover, a static capacity makes it easier to understand the often complex interaction of RADICCO multi-level control loops and the CCAs' control loops.

Applying constant delay is a third simplification. Although there are close to zero queues in today's transport and core networks and routes usually are static during a transmission, the absolute delay typically varies anyway. One cause of this delay variation, also called jitter, is found at the senders themselves: Although the CC design assumes OSs sending each packet independently, this is not necessarily the case for modern computer architectures anymore. To save computational cost caused by today's high packet rates, offloading mechanisms [168, 169], e.g. Generic Receive Offload (GRO), TCP Segmentation Offload (TSO) and checksumming, as well as interrupt saving mechanisms [170, 171], often called interrupt coalescing, have been

developed and deployed. The effects off the offloading mechanisms, often generally referred to by TCP Offload Engine (TOE), on TCP performance have been evaluated [170, 172, 173] as well as the effects of interrupt coalescing [174, 171]. It was found that both mechanisms may affect the operation of TCP negatively but for most scenarios these drawbacks seem to be more than compensated by the benefits since both mechanisms are widely in use today.
Nevertheless, modeling their effects is far from trivial and error prone. To our knowledge, there is no evaluation of an algorithmic proposal regarding TCP or CC incorporating the effects of offloading or interrupt coalescing. This also applies to research on delay-controlled CCAs where the impact is expected to be more significant than for loss-controlled CCAs. In case of our evaluation of RADICCO, the static delays remove dispensable variation in our measurements and therefore makes interpreting the measurements easier and keeps this thesis focused. On the one hand, configuring constant delays implies a risk for synchronization of the different control loops of RADICCO and the senders' CCAs. On the other hand, using constant delays means that our results are immediately comparable with other results published in the TCP and CC research community.
Based on these arguments, we decided to apply constant delays without jitter to our simulated links.

4.2.3 Scaling Load

Since RADICCO changes the scheduler's operation only during overload, our simulations focus on overload situations. Overload in this context means a situation in which there is a higher offer of elastic load than can be transported. Nevertheless, the load is elastic, i.e. controlled by transport layer CCAs, so it adapts to the capacity shortage.

To evaluate the effectiveness of RADICCO we need to scale the extent of overload while maintaining an otherwise equivalent simulation scenario. This overload should roughly correspond to the depth of the dip in performance today's subscribers often see when measuring their Internet access speed during peak periods (see Section 2.1.3 and Figure 2.3). This extend of underperformance, i.e. the difference between access link capacity and allocated bandwidth directly corresponds to the maximum benefit that can be achieved by RADICCO since this headroom is the maximum that it can additionally allocate to that subscriber.

Scaling overload in simulations is a non-trivial task however. The crucial point is that the statistical evaluation of simulation results is valid only when evaluating steady state systems. This steady state may refer to different levels of evaluation: On the one hand packet level, or on the other hand application level. On packet level, overload risks functioning of the overall system and therefore in today's networks is completely avoided by the fact that most traffic is congestion controlled (see Section 2.3.3). Regarding our simulations, every simulation according to the presented model results in a steady state on packet level since we use CCs and elements of limited sizes only, in particular buffers and links. So, any measurements on packet level have stable stochastic properties.
In contrast, application level overload in real networks can be easily tolerated since the CC active at the end hosts prevent this overload from bringing harm to the overall system. Therefore, overload on that level means demands piling up, i.e. transfer requests cannot be completed until the next demand arrives. In reality, typically there is a higher-level control, e.g. a human user, that

intervenes and reduces application level demands, e.g. by canceling a session. In our simulations, in particular statically defined traffic models, e.g. based on Inter-Arrival Times (IATs), may result in stable as well as unstable systems on demand level: The system is stable if on average demands can be transported, and unstable if the demands exceed the average capacity available to the respective sender. In simulation, such unstable systems show a longer mean transfer time than the mean IAT. We use this fact to identify configurations that result in unstable systems, see also Section 4.5. If the traffic model is elastic, e.g. arrivals are based on delay from completion of the last transmission, the system cannot be unstable on the application level.

When designing our simulation parameterizations for RADICCO's evaluation, we aim to provide global control for quantifiable overload on packet level as outlined above, i.e. in terms of offered stochastic load. In general, there are several options to achieve such global control:

- By scaling the number of subscribers.
- By scaling the access link capacity.
- By scaling the traffic model, i.e. the size and/or arrival of transmitted objects.
- By scaling the bottleneck links, i.e. capacity of the BNG interface and/or of the aggregation switch. interfaces

We decided to scale the bottleneck links. There are good reasons not to choose one of the other options as argued in the following.

If we would change the number of subscribers, both the number of arrival processes and the number of CC control loops are changed. So, we introduce a change both on packet level as well as on demand level. For the possible numbers of subscribers in our simulations, both changes typically heavily impact the overall system behavior. Moreover, adapting the number of subscribers allows only few steps, thus with very different relative changes.

Scaling the capacity of the access links would only be desirable if the offered load scales proportional with the access link speed. This is obviously not true for all static traffic models. Regarding elastic models, there are models that scale load about linearly with access link capacity, e.g. the load offered by greedy connections scales about linearly with the access link capacity. But this is not true for most other elastic traffic models: For instance, for delay-based traffic models the offered load scales somehow with the access link speed, but far from linearly. For this work, we only consider distribution-based traffic models, which means we do not consider traffic models using dynamic calculations. Such model would also modeling the dynamics of DASH-like VoD, but we consider such complex traffic models out of the scope of this thesis.
So, scaling the access link capacity would require to also adapt the definitions of traffic models, which is not the desired simple control parameter. Moreover, even if this adaptation would be perfect, it results in different behaviors of the CCAs since the cwnds grow to different maximums (assuming the maximum rate allocated by the scheduler is relative to the access link bandwidth). But smaller congestion windows mean lesser control loop updates and therefore slower reaction to changing environments.

Scaling the load by scaling the size and arrival of transmitted objects again rises concerns since it leaves the ratio between feeding capacity and maximum receiving capacity the same.

Nevertheless, we consider this relation crucial to vary since it also defines the possibilities of reallocating bandwidth.

Therefore, we consider scaling bottleneck links the best option. Nevertheless, the operation of CCAs is always based on discrete events, e.g. packet arrivals. When scaling load in such systems, effects typically are not strictly linear, e.g. with less bandwidth available to a foreground TCP connection the transmission of the same amount of data causes higher absolute packet loss and retransmissions.

For the simulations presented in this chapter, we varied the bandwidth of both the BNG interface and the aggregation switch interfaces, i.e. the capacities of both the scheduler's root node, C_R, and of its edge nodes, C_e. To provide a basis for comparing the load on these bottleneck links between simulations using different traffic models, we used a two-level scaling of these capacities:

1. We calculate the overall stochastic load as sum of mean stochastic loads of all subscribers.
2. We overload each aggregation level by configurable factor. For that, we define overload factors for the BNG interface, OLF_{BNG}, and for the AGS interfaces, OLF_{AGS}.

For the first step, we independently measured for each traffic model TM the long-term average link layer load $L(TM)$, that occurs if the subscriber's access link is the bottleneck. It considers the layer two, three and four protocol headers overhead (for TCP over IPv4 over Ethernet 66 Byte per 1448 payload, so 4.558 %) but also overhead caused by unnecessary retransmissions. The load $L(TM)$ therefore not only depends on the traffic model, but also on the capacity of the link, the RTT and the buffer at the bottleneck and, if applicable, its AQM. Therefore, we left these parameters unchanged for all presented simulations to provide comparable results. Nevertheless, it is crucial that first, the evaluated overload periods result in reduced and varying available bandwidth, so the actual effective link layer load will probably increase due to additional retransmissions. Second, $L(TM)$ is just an average of in most cases a distribution-defined pattern of transmission and idle phases. Applying these stochastic patterns to several subscribers in an access network causes loads at aggregation links to vary and sometimes even the location of the bottleneck to vary over time: There are phases of heavier overload at aggregation interfaces than the calculated average but also phases of the aggregation interfaces not being fully loaded so that the access links constitute the bottlenecks.
When configuring a specific traffic model TM for a subscriber i, j in a simulation scenario, we also assign the respective average load $L_{i,j} = L(TM)$ to that subscriber. Based on these subscriber loads we calculate the capacity of the AGS interfaces and then the BNG interface capacity.

For the second step, we use the overload factors OLF_{AGS} and OLF_{BNG}. The AGS interface capacity C_e is defined as given in Equation 4.1.

$$C_e = \frac{\sum_{all\ leaf\ nodes\ i,j} L_{i,j}}{\#access nodes * OLF_{AGS}} \tag{4.1}$$

This definition means that the expected average load of all subscribers is divided by the number of ANs and by the configured overload factor for AGS interfaces. It configures the same bandwidth for all AGS interfaces since this probably reflects reality in hierarchical access networks. This calculation does not take into account any unbalanced load distribution among the ANs caused by subscribers receiving traffic according to different traffic models. Again, this reflects the

situation in real networks: The operator just knows the number of subscribers but cannot foresee their behavior.

The BNG interface capacity C_R is defined as the sum of all AGS interface capacities, i.e. C_e times the number of ANs, divided by its configured overload factor OLF_{BNG}. The calculation is also given in Equation 4.2.

$$C_R = \frac{\#access\,nodes * C_e}{OLF_{BNG}} \tag{4.2}$$

In the presented simulations, we only adapt the stochastic overload of either BNG or AGS level: According to the targeted evaluation of the different load shifting mechanisms, we induce statistical overload at the AGS interfaces only in case of the BROAD topology and at the BNG interface only in case of the DEEP topology.

Therefore, we define the overload factor OLF as the product of the two level specific overload factors as depicted in Equation 4.3.

$$OLF = OLF_{BNG} \cdot OLF_{AGS} \tag{4.3}$$

The overload factor OLF therefore provides the desired single parameter to scale the overload. More precisely, the OLF defines the relation between offered load and available bottleneck capacity.

A final remark:
Since the SimLib [158, 159] and more importantly the Linux-internal time management work with nanosecond resolution, we are often not able to use the exact calculated capacities, i.e. we do not model exactly the desired overload. If the total load changes by parameter variation during one set of simulations, this leads to slightly differing sets of simulated load levels as for instance in the simulations with rate-limited traffic as can be seen in Figures 4.27 to 4.32.

4.2.4 Traffic Models

Internet traffic is continuously changing in general, both because the users change their behavior as well as because service providers create new services and update and re-design existing ones frequently. In consequence, there is no value in aiming to capture today's traffic in detail since it will be a different composition of different services tomorrow. Nevertheless, it is possible to define fundamental classes of services, derive their fundamental behavior and type of QoS requirements.

With regard to RADICCO's goals and the traffic evolution until today, we consider the following traffic types being of special interest:

- Greedy transmissions
  since this is the most basic model and may model permanent background transmissions
- Otherwise rate-limited long-term traffic
  since it is the traffic we expect RADICCO to perform worst

- Downloads of software updates
  since this traffic is typical background traffic and more challenging than greedy transmissions.
- DASH-like VoD Streaming
  since such traffic makes up for ~60 % of peak period traffic
- Web Browsing
  since it is crucial for the users' perception of service quality

The motivation for using these models defines two groups of traffic models: The first two traffic models are artificial models, they are not aligned to specific services. The second group of three traffic models aim to model services that we consider important.

In a simulation, each subscriber is assigned a traffic model, and all transfers defined by the traffic model are transferred using a single transport layer connection. For each traffic model, we will provide background information and, if applicable, the parameters of our respective traffic model.

4.2.4.1 Greedy Transmissions

Greedy transmissions are transmissions that are never limited by the sender application, resulting in a transmission governed only by the CCA and the network performance (as perceived by the CC). Although real transmissions do have some specific size, i.e. stop after a specific volume, not only very large transmissions may be approximated by this concept. They are suited to represent traffic consisting of a varying number of overlapping TCP connections of limited life time (possibly even of several subscribers) that in total consume all available bandwidth. Greedy transmissions may therefore be used to model P2P traffic. But also in general greedy traffic should mostly be background traffic due to its elasticity and hence obvious tolerance in completion times. We do not show results of simulations with greedy background transmissions and rather show results on more challenging and critical traffic models.

4.2.4.2 Otherwise Rate-limited Long-term Traffic

Background

The traffic examined in this scenario does not represent a specific service present in today's Internet. We include this traffic model since this traffic is the most critical type of traffic for our approach. As explained in Section 3.8.5.3, using RADICCO is expected to result in oscillating traffic type recognition for such traffic.

Every false recognition as background subscriber results in a reduced bandwidth assignment compared to the correct foreground recognition. If at some point of time the respective incoming rate is higher than the assigned reduced rate, a queue starts building up at the BNG, resulting in additional and usually undesired delay and delay variation. For newly starting foreground flows with the bottleneck at the BNG scheduler a false recognition as background has no strong impact

for three reasons. First, this usually happens just once at the start of a transmission. Second, the rate reduction is minor. Third, the false state, and thus the rate disadvantage, lasts only for a short time and usually is even left within the slow start phase, which allows seizing bandwidth that becomes available after correct recognition very fast. However, for transmissions that are limited at another place before the access network, these statements do not necessarily hold. Therefore, for this type of traffic we expect large packet delays.

The place of limitation may be located somewhere in the network between sender and BNG, but also in the sender, which most prominently applies to real-time streaming services such as third-party VoIP or real-time video conferencing. Nevertheless, all this only applies to traffic that is not prioritized by the ISP. Typical examples are Skype or Web Meeting conferences at private subscriber lines. For such traffic, low packet delay is crucial for QoE. While such rate limits may affect any type of traffic, we examine the impact of RADICCO only for greedy rate-limited traffic since we just aim at providing a first glance on the severity of the expected deficiency.

Traffic Model Definition

To model long-term rate-limited traffic we use a greedy TCP Cubic connection and configure the respective SenderPacingLink with the lower capacity of the rate limit. The SenderPacingLink represents the network interface that connects the sender Linux stack to the simulated network as shown in Figure 4.3. Note that the SenderPacingLink is equipped with 100 ms of buffer.

4.2.4.3 Software Updates

Background

The prevalence of so-called app stores on many platforms, e.g. Apple iOS, Google Android, Microsoft Windows and Apple MacOS, facilitates and fosters frequent software downloads for a wide range of purposes, not only for fresh installations but most prominently for security and function updates. Such software updates are often downloaded and installed automatically when becoming available without user interaction. Similar mechanisms are also available for and applied to OS updates, e.g. for iOS, Windows, MacOS and Linux distributions. Most software downloads are not linked to user interaction, but are downloaded and installed by automated procedures. Such transfers should therefore be background traffic and Microsoft as well as Apple report to implement measures to achieve such behavior, Apple indeed using the LEDBAT CC [114].

The size of software updates is constantly increasing, and covers a wide range from few megabytes to several gigabytes (recent examples from late 2016: Microsoft's "Windows 10 Anniversary Update" is about 3 GB, Apple's iOS update to 10.0.2 is about 2.2 GB). Nevertheless, regarding performance evaluation, these huge downloads provide no further insights since they resemble a greedy transmission most of the time. Nevertheless, most software updates are significantly smaller. We are not aware of data on the download sizes, but they are changing fast anyway so we use a reasonable, but arbitrary and simple traffic model. For the evaluation of RADICCO, the overall behavior is more important than the size distribution: In many cases

software updates are downloaded and installed in turns, resulting in a long download phase, followed by a phase of inactivity from the network's perspective.

Summarizing, automated software updates are a predestined candidate for CC managed background traffic and are responsible for large and increasing amounts of traffic. Due to their behavior, they are a more interesting traffic model than trivial greedy background transmissions.

Traffic Model Definition

We use a delay-based traffic model that uses uniformly distributed waiting times and object sizes. In detail, the waiting times are uniformly distributed between 8.5 and 11.5 seconds and object sizes are uniformly distributed between 30 MB ($30 \cdot 10^6$ Byte) and 300 MB. Since RADICCO benefits from more stable load situations on any aggregation level, the object sizes are chosen rather too small than too large to provide realistic and not overly optimistic results.

4.2.4.4 Video-on-Demand Streaming

Background

Today, VoD streaming service use DASH or similar techniques. This type of traffic accounts for the biggest share of peak period traffic, e.g. in total more than 60 % in North America [85] and at least about 40 % in Europe [83], see also Figures 2.10a and 2.10b. A VoD streaming session consists of one or few TCP connections that transmit video data in bursts to keep the playback buffer at the playback device sufficiently filled. When the playback buffer runs low, the VoD receiver signals the sender to reduce bandwidth consumption. The sender can then step down on a so-called "bitrate ladder" [79], i.e. switch to transmitting the video in a lower resolution or lower quality encoding. Any step down in bitrate results in a decrease in QoE but usually the switch can be handled without stalls, so the worst case QoE impairment is avoided.

There are several publications on the properties of DASH video streaming [99, 175, 176], and we also conducted measurements for the Netflix, Amazon video and YouTube services ourselves. We analyzed four streaming sessions in detail and found three being streamed from data centers near Frankfurt, one from the USA near New York. All streams used several—most of the time three—concurrent TCP connections. The findings of our analysis differ significantly from the ones given in [99] and [176] and other non-academic analyses. This underlines the high volatility of traffic characteristics for this particular traffic. Obviously VoD streaming traffic varies both from service provider to service provider as well as over time, therefore we only aim at correctly modeling the fundamental properties.

We performed a quick analysis, which we do not include here because of its minor importance and space reasons. The results showed that for three streaming sessions the IATs of traffic bursts follow a rather narrow distribution around two seconds. Our simple analysis (based on a fixed idle-time threshold of 250 ms) also resulted in some outliers, most of them at about four and some at six seconds IAT. For a fourth session, we found a very narrow IAT distribution with the sharp peak at ten seconds. For all sessions, the burst sizes are much bigger than the burst

sizes reported in other papers. Probably the reason is an increased resolution, all our streaming sessions used the Full High Definition (FullHD) resolution. The distributions of the burst sizes were very different, some showing rather uniform distribution between 1 MB and 3.5 MB while others show a significant peak at about 2.5 MB, both combined with the short (about two seconds) inter arrival intervals. These measurements make clear how volatile the exact properties of VoD streaming traffic are. Nevertheless, we identify several stable properties of today's DASH-like VoD streaming:

- Data is sent in bursts.
- The bursts are sent in rather fixed intervals.
- The volume per burst depends on video properties such as resolution, frame rate, quality and video content.
- If several bursts are not received in time, the playback buffer decreases.
- If the playback buffer falls below a certain threshold, the server reduces the video bitrate, i.e. QoE decreases.

Traffic Model Definition

For our analysis, we do not aim at modeling the control mechanism for the bitrate selection. In contrast, we aim to examine which conditions suffice to successfully deliver the best quality service and therefore model maximum quality video streaming traffic as non-adaptive streaming traffic. In these models, the objects to be transferred correspond to the traffic bursts in the measurements.

More precisely, we use two models for DASH-like VoD streaming traffic inspired by the two patterns we saw in our own measurements. Both traffic models are based on a narrow normal-distributed IAT and a rather wide-spread uniform distribution of object sizes, i.e. of the independent data bursts transmitted during a VoD streaming session. These IAT models are guided by our impression that technically, bursts are triggered in fixed intervals but random delays at the sender, e.g. at its storage back-end, cause the actual intervals to vary a little. We choose rather large object distributions we saw only for an older movie which shows much noise. Nevertheless, we aim at modeling a rather challenging situation and such model might even fit for streaming higher resolution but low noise material.
The first model called short interval model uses short intervals and small objects. It is defined by a normal-distributed IAT with a mean of two seconds and a standard deviation of only 0.1 seconds. We do not allow IATs to be lower than 0.5 or bigger than 3.5 seconds. The object sizes are uniformly distributed between 1 MB and 3.5 MB. The second traffic model called large interval model is based on longer IATs and bigger objects. The normally distributed IATs have a mean of ten seconds and again a standard deviation of 0.1 seconds. We clip the distribution at 8.5 and 11.5 seconds. The object sizes are uniformly distributed between 5 MB and 17.5 MB.

4.2.4.5 Web Browsing

Background

Web browsing by now does not account for a large share anymore but is used by virtually all users and typically a human user waits for the result. Therefore, from the user perspective it provides an instant information on service quality. From the ISP's perspective, delivering a bad service quality for web browsing poses the risk that the user generally associates a bad performance with that ISP. Today's typical web pages consist of many objects, often retrieved from different servers. If a web browser does not use a proxy, it therefore sets up many TCP connections to request and receive the content. We use a simplistic model here: We do not model the request and model a web browsing transfer as one download representing the total of all the object transmissions. Moreover, we use only one TCP connection to receive all web browsing transfers. Due to the long reading time in a web browsing model, this connection resets the cwnd to the initial window and carries out slow start for each object transfer. Nevertheless, it does not require a new three-way-handshake for a new transfer. Modeling web traffic is challenging and there exist many publications on that topic (from [177, 178, 179] to [180, 13, 181]). In general, web traffic is modeled by a (very) heavy tailed distribution for the object sizes and another distribution for the reading time that follows each transfer.

Traffic Model Definition

For the object size distribution, we chose to scale the model of Hernandez-Campos [180] to a mean object size of 1 MB which is often reported to be today's typical website size and about matches the findings of Pries et al. in [181] (833 kB mean total size in 2012). Since we are not interested in the large transmissions already covered by the software updates traffic model, we cut the object sizes at 30 MB, the minimum object size of the software updates traffic model. For the reading time distribution, we adopted the log-normal distribution proposed by [182, 181] ($\mu = -0.495204$, $\sigma = 2.7731$, cut off at 10.000 seconds).

4.2.5 Model Parameterization and General Simulation Parameters

There are several basic parameters such as buffer sizes, AQMs or link delays that can be tuned in the described simulation model. We use a uniform setup for all simulations to achieve comparable results. We use a simple tail drop queue since AQM tuning is a delicate task and its effects may blur the view on RADICCO behavior. We use buffer sizes worth 100 ms of continuous transmission of the respective link, following the argumentation given in Section 2.4.4.

We also use an RTT of 100 ms—by setting the delays of the access links only to 50 ms, parameter $\$OWD$ in Figure 4.3—although many of today's connections show shorter RTTs. Lower RTTs result in faster control loops of the involved CCAs, so a large RTT represents a worse case for RADICCO. In fact, simulations with lower RTTs show higher utilization since the involved CCs can seize freed bandwidth faster. Since there are no other effects of special interest to be seen in these simulations, we do not show these results.

We use 20 Mbit/s downstream capacity and 5 Mbit/s upstream capacity for all access links, a bit more than the 14.1 Mbit/s average downstream speed reported by Akamai for Germany in 2016 [6].

We carried out all simulations for at least 2.000 s simulated time after a transient phase of 15 s allowing the system to reach steady state. For statistical evaluation, we partitioned every simulation into ten batches if not stated differently.

4.2.6 Algorithmic Parameters

For the results presented we did not change any internal algorithmic parameter of RADICCO but used the values presented and argued in Chapter 3, Sections 3.7 and 3.8.

We use

- A relative background buffer usage threshold of 0.35, i.e. $tt\_recog^{relQThresh} = 0.35$
- A buffer observation period of 2 s, , i.e. $tt\_recog^{obsDuration} = 2\,\mathrm{s}$
- A minimum background target rate of 1 Mbit/s, i.e. $bg^{min} = 1\,Mbit/s$.
- A per-packet target rate decrease of 37.5 kbit/s/packet , i.e. $\beta_{reduce} = 37.5\ \frac{kbit}{s*packet}$

4.2.7 Reference Scheduler Implementation

We implemented the WF^{2}Q+ packet scheduler and enhanced it by rate shaping functionality. We use this implementation as reference scheduler since it provides a *O(1)* GPS-relative delay, proportional fairness and an *O(1)* nWFI, so the best-possible scheduling according to all standard scheduling metrics except computational complexity. For simulations, the complexity is not of primary importance since we aim to show the potential of the approach. Moreover, for the small topologies in our simulations, WF^{2}Q+'s complexity of *O(log n)* would even be well acceptable in real implementations ($\lceil log_2 20 \rceil = 5$).
For all simulations, we configure all per-subscriber queues as drop-tail queuing disciplines of 250 kB of buffer, equivalent to a maximum queuing delay matching the configured RTT of 100 ms.

4.3 Performance Metrics

We evaluate performance regarding the four domains listed as goals and moreover, we present core data on RADICCO's internal state.
First, we examine if the user in the examined scenarios benefits from RADICCO (Quantitative Objective 1), i.e. if RADICCO is likely to induce an increased QoE of the service the foreground traffic belongs to. With that regard, we do not aim to produce specific QoE measurements; this is out of the scope of this work. Nevertheless, for each traffic type examined in our simulations the QoE primarily depends on one or two specific QoS properties such as object transfer time. We

will discuss these dependencies in Section 4.3.1 and accordingly present selected measurements for every scenario. Second, we evaluate the bottleneck utilization (Quantitative Objective 2). This measurement indicates a possible price for any gains in QoE in terms of wasted resources. Third, we examine achieved fairness among the foreground subscribers (Quantitative Objective 3) and among the background subscribers (Quantitative Objective 4). Moreover, we will present data on correctness of RADICCO subscriber recognition where suitable.

4.3.1 Improved Quality of Service for Foreground Traffic

We assume that only the foreground traffic contributes to immediate QoE, so we only evaluate the QoS of foreground traffic. Generally, by design of the algorithm any subscriber consistently recognized as foreground cannot be adversely affected by the proposed algorithm. But such consistent foreground recognition cannot be guaranteed and is even unlikely due to the design of the background detection algorithm for many traffic types in the real Internet and our respective models. Therefore, we present the crucial QoS criteria for all traffic models used in the simulations.

Greedy transmissions only aim at transmitting as much data as possible, so this represents the only criterion for QoE in that case.
Otherwise rate-limited long-term traffic is different compared to all other traffic types since we do not make a specific assumption on the service. VoIP traffic, and with restrictions also video conferencing traffic, may serve as example for demanding traffic with such properties. At least for VoIP, the QoE-relevant unit is the single packet. For the transmissions of single packets the mean delay as well as the jitter are crucial properties since they define the overall playback delay that should be below the International Telecommunication Union (ITU)'s threshold of 150 ms for one way delay [G.114]. So, for otherwise rate-limited traffic we present mean packet delay as well as maximum packet delay.
Software updates are useful only after completion, so the crucial measurement to assess QoE is the mean transfer time.
For our static model of DASH-like VoD streaming any impairment of QoE would be caused by the playback buffer running empty. This happens if an object's transfer is longer delayed than the playback buffer can compensate. Since playback buffer management is out of the scope of this work, we assume a large playback buffer. This allows to judge on QoE by measuring the mean object transfer time only: If the mean object transfer time is smaller than the mean object IAT, we can assume the large buffer to compensate any temporary delays.
For web browsing, the same reasoning as for software downloads applies.

4.3.2 Bottleneck Utilization

The managed scarce resource should not be wasted, so a high utilization of the managed BNG link is desirable. In all simulations, the bottleneck BNG link is overloaded on average, i.e. on average there is more traffic offered than can be transported. Nevertheless, there are several reasons for idle times of bottleneck link.

All scenarios base on stochastic traffic models, so the number of active subscribers is a random variable. If this number is small at some point in time, e.g. after one or several transfers have been completed and the respective subscribers became inactive, there are several reasons for the remaining transfers not being able to fully utilize the BNG downstream interface.
First, the number of remaining active subscribers may be so small that the access links become the bottlenecks and limit the throughput of the BNG interface. This only happens at low stochastic overload, i.e. for low overload factors.
Second, even if the number of remaining active subscribers is sufficient, the reduction in load results in an increase of rates allocated to the remaining active subscribers. The CCA control loops may be too slow in increasing their cwnds to keep their access link queues from running empty and thus the respective subscriber becomes inactive for some time. At this point the existence of buffers at AGS and AN interfaces and their usage by the packet scheduler plays a role: If these interfaces are equipped with buffers, the scheduler may bring forward other, non-empty queues that would be not yet eligible in terms of strict maximum rate control since their respective links are still busy transmitting the last (or even an earlier) packet. Our WF^{2}Q+ scheduler implementation only uses a buffer of one packet at these interfaces. Thus, it may induce idle times in situations, in which a DRR scheduler would just rotate on and fill buffers at these interfaces. It is not obvious if such a system would result in improved utilization or just delay the idle times. Anyway, since the reference HFS implementation is the base scheduler of our RADICCO implementation, measured differences in utilization base on the same behavior and thus indicate an impact of RADICCO.

The reduced target rates of subscribers recognized as background subscribers by design cannot directly cause underutilization since the effective rates always utilize any bandwidth that would be wasted otherwise. Nevertheless, the CCAs of subscribers having recently been served with reduced rates may have even more difficulties to increase their cwnds fast enough. Moreover, the general approach of RADICCO of adapting rates results in more rate changes than inflicted by the changing load and may amplify these. Any of these additional rate changes comes with the risk to cause underutilization as detailed above.

All the reasons for underutilization during stochastic overload discussed above may be seen as results of stochastic processes. Obviously, the probability of all these reasons decreases with increasing number of flows. So, for scenarios with more subscribers, the described reasons for underutilization will less likely become effective, i.e. the mean utilization at the same overload factor increases. Consequently, regarding utilization our topologies represent a rather bad case for RADICCO.

Independently, high utilization is only a secondary goal. A lower bottleneck utilization may very well be acceptable if the QoE of the foreground traffic is increased.

4.3.3 Fairness of Bandwidth Allocation

As stated in our goals, see Section 3.4, there is a significant difference in importance between fairness among foreground subscribers and among background subscribers. Nevertheless, the metric used to measure this fairness is the same, therefore we discuss both in this section. With today's CCAs, fairness can only be demanded from a scheduling algorithm for same external

parameters such as used CC, RTT and of course offered load. For our simulations, this is the case except for the load: We use stochastic load. Nevertheless, we need to compare medium-term averages anyway since CCAs such as TCP Cubic do not converge but achieve long-term fairness (see Section 2.4.2 and [106]). The durations simulated for this evaluation should be sufficient to provide realistic fairness measures for the stochastic traffic models used.

To evaluate the intra-class fairness of bandwidth allocation, we use Jain's fairness index [183], a fairness metric widely used in TCP-related research. Regarding this metric, a value of 1.0 indicates perfect fairness. The worst-case fairness depends on the number of competing entities, i.e. subscribers in our case. For N competing entities the value $\frac{1}{N}$ indicates worst case fairness, i.e. one subscriber receives all bandwidth.

By design, RADICCO does not manipulate weights of subscribers recognized as foreground subscribers, thus in case of perfect foreground recognition, RADICCO results in the same fairness as the base HFS. Nevertheless, all non-continuously active subscribers will be recognized as background subscribers when a new transfer begins after a sufficiently long idle time. Such temporary false recognitions can be the source of unfairness if they, i.e. their durations, are not equally distributed among subscribers.

In contrast, there are several sources for unfairness among background subscribers: First, as for foreground subscribers, the durations of the respective recognitions may be unequally distributed. Second, the dynamic rate adaptations of RADICCO may amplify short-term random unfairness. And third, the control loop of RADICCO and the delay-controlled control loops of the background CCAs may interact unfavorably and further amplify unfairness: An initial unfairly low rate induces delay, to which the CCA reacts by reducing the sending rate, which makes RADICCO continue to recognize the subscriber as background and further reduce its target rate.
Therefore, a lower level of fairness is expected among background subscriber than among foreground subscribers. Nevertheless, fairness among background subscribers is of lowest importance since the respective traffic does not contribute to QoE.

Since in unbalanced multi-level topologies fairness as a secondary goal is traded off for a more desirable bandwidth allocation (see also Section 3.4), we do not evaluate fairness in the DEEP topology.

4.3.4 Correct Subscriber Recognition

To understand performance results, or why RADICCO results in certain effects, it is helpful to gather statistics on its internal states. For RADICCO, there are two crucial internal states: First, the effective rate assigned to a subscriber or to a set of subscribers. This is already indirectly reflected by the QoE measurements. Second, the recognition of subscribers. We will show the data on subscriber recognition as the only data on internal states, since it supports understanding the reasons for RADICCO performance.

Regarding the correct identification of traffic there are several dimensions to be considered when assessing the impact on RADICCO performance. First, it is more important that foreground traffic is recognized as such than that background traffic is correctly identified: Failing the first

may result in RADICCO negatively impacting the crucial foreground QoE while failing the second only diminishes the benefit gained by RADICCO. Second, the distribution of the duration of false recognition is crucial since the effect of a subscriber being recognized as background is negligible at the start of the rate adaptation and grows over time to some maximum extent (within the first few seconds, depending on the effective rate). This again applies to both, foreground and background, but again the severity differs. For foreground subscribers, short periods of false recognition results in a barely reduced bandwidth for that period only. So, any number of short periods of false recognition is acceptable for foreground, while a longer time of false recognition results in a severe reduction of allocated bandwidth and, depending on the service transported, QoE degradation. In contrast, if false recognition time of background subscribers is distributed in many short periods of time, every such event will reset the target rate to the maximum rate, so RADICCO will effectively not retrieve significant bandwidth from such background subscribers.

Nevertheless, even duration and number of phases of false recognition are not sufficient to estimate the extent of the impact on QoE: If there is low overload, even recognizing foreground traffic falsely as background may cause no substantial harm: RADICCO does not simply apply the calculated reduced target rates for background subscribers but tries to exploit remaining bandwidth instead. Therefore, a background subscriber's effective rate may be much higher than his target rate.

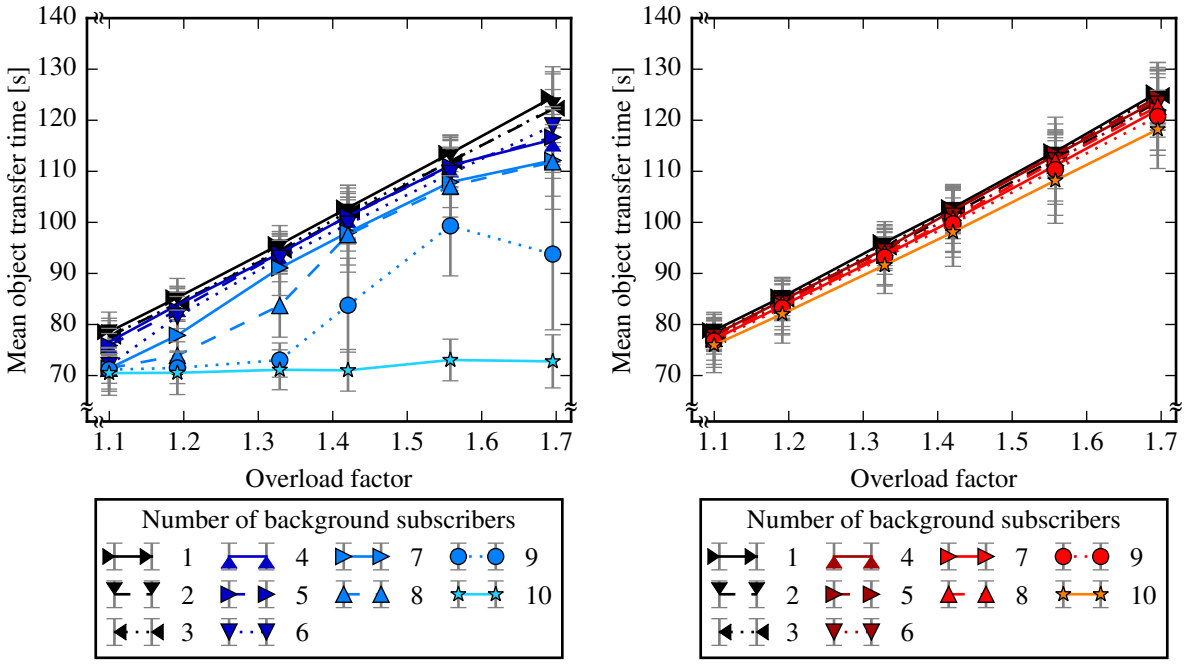
In the following, the terms "background subscriber" and "foreground subscriber" refer to a subscriber's configured traffic type, not the traffic type recognized by RADICCO.

4.4 Performance for Software Updates Traffic

In this scenario, we examine the effectiveness of prioritization by RADICCO when downloading sequences of rather big objects, e.g. software updates. Such transmissions may be triggered by automatic processes but also by explicit request of a user. So, in the first case, the process runs in background and there usually is no decrease in QoE in case of delayed completion times. In the latter case in contrast, the process runs in foreground and the user may wait for completion, so a delayd completion may result in a decline in QoE. Accordingly, the first group should be—and partly is already today—using a background CC and the second a foreground CC. We model such situations in this set of simulations.

In this scenario, we simulate twenty subscribers, among these one to ten background subscribers. These subscribers are equivalent to an average background share of 5 % to 50 % in offered load since both types of subscribers receive traffic according to the same traffic model. Nevertheless, this simple equation only holds for the long-term average and moreover is not applicable to all other scenarios. Therefore, we state the number of background subscribers rather than the average share in offered load in the following.

We decided to show rather all values for the presented evaluations, e.g. for all simulated numbers of background subscriber, than only selected subsets since many results show non-continuous effects. So, showing results for selected values implies a risk to distort the overall impression.



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.4: Foreground object transfer times for the BROAD topology, the traffic model of software updates and TCP Vegas-controlled background traffic

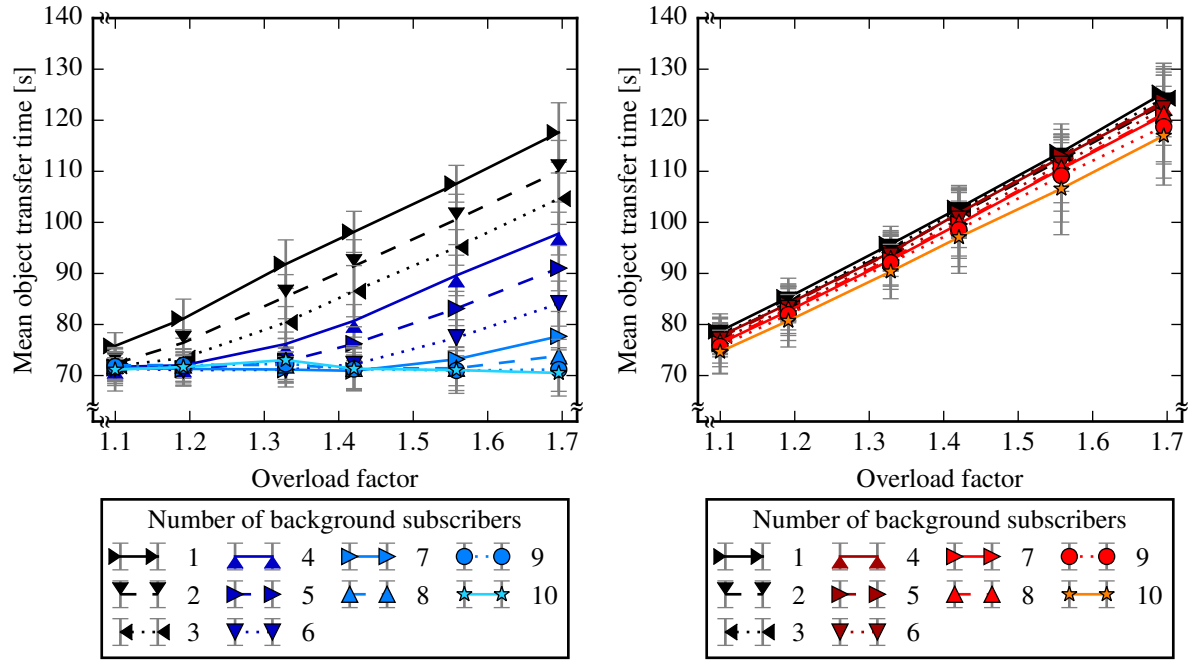
4.4.1 Transfer Times of Foreground Traffic

Object transfer times consist of the transmission time of the data of that object and a waiting time until the transmission is started. The transmission time is always greater than zero, and the waiting time is zero or greater than zero. For this scenario, we evaluate the mean transfer time since every transmission results in a distinct QoE, so the overall QoE is directly correlated with the mean transfer time.

We evaluated the foreground object transfer times for the BROAD topology as well as for the DEEP topology. For the latter, we only show results for the clustered distribution of background subscribers. The clustered distribution is characterized by the fact that there is at maximum one access node serving background and foreground subscribers, while all other access nodes serve only one type of subscriber. This forces RADICCO to adapt rates on the AGS level, i.e. to adapt the allocations for mixed aggregates served by the root node scheduler. Moreover, the clustered distribution is the worst-case of a distribution of subscribers for RADICCO, since it as far as possible prevents adapting rates on AN level, i.e. at the edge node scheduler.

BROAD Topology

For the BROAD topology, Figure 4.4 shows the mean foreground transfer times in case of TCP Vegas-controlled background traffic, while Figure 4.5 shows the mean foreground transfer times in case of uTP-controlled background traffic. Regarding the transfer times achieved by the reference HFS, there almost is the expected linear relation between load and transfer times as



(a) Applying RADICCO

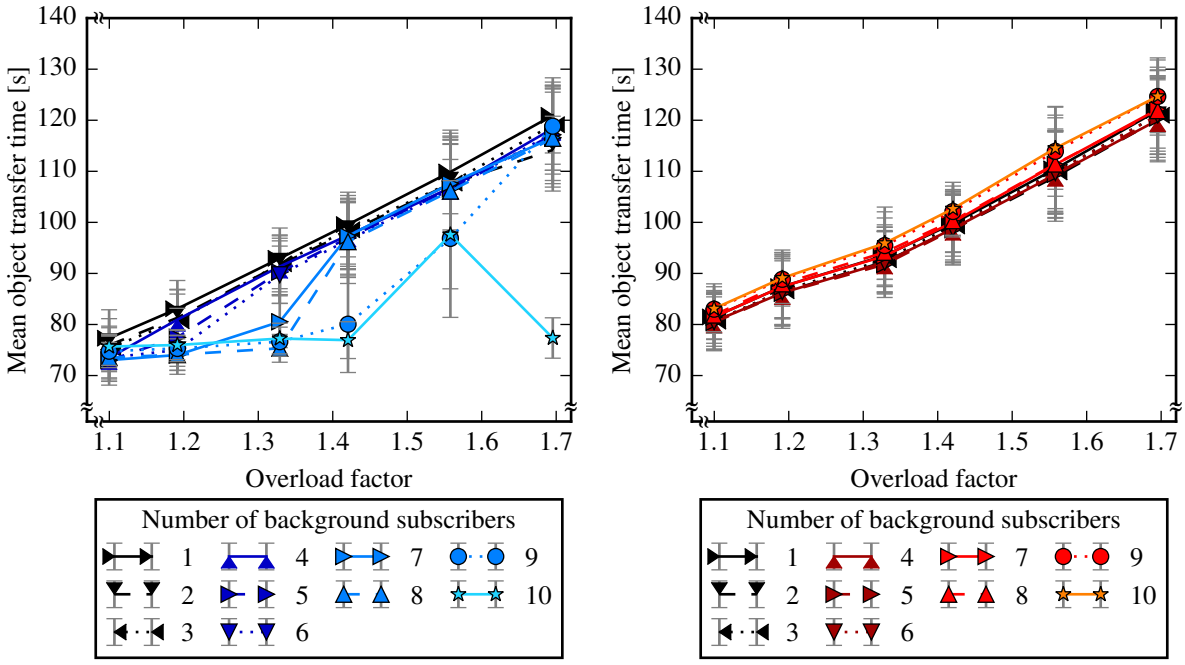
(b) Applying the reference HFS

Figure 4.5: Foreground object transfer times for the BROAD topology, the traffic model of software updates and uTP-controlled background traffic

visible in the Figures 4.4b for TCP Vegas-controlled background traffic and in Figure 4.5b for uTP-controlled background traffic. In both cases, there is slight improvement, i.e. a reduction of transfer times, with increasing fraction of background subscribers. This is caused by these protocols seizing bandwidth more carefully and slower than TCP Cubic, the CCA replacing the respective background CCA. In consequence, if the bandwidth available to a transport layer connection is not constant, as in our simulations, on average the rate of a background transmission is slightly lower than that of a foreground transmission of the same object size.

When assessing the transfer times achieved by RADICCO, shown for TCP Vegas-controlled background traffic in Figure 4.4a and for uTP-controlled background traffic in Figure 4.5a, there is a benefit of RADICCO detectable for almost all cases. The extent of this benefit, i.e. the reduction in foreground transfer times, and its evolution depending on the amount of background subscribers differs heavily depending if the background traffic is controlled by TCP Vegas or uTP. For TCP Vegas-controlled background traffic there is a remarkable irregularity for nine background subscribers and overload factors 1.56 and 1.69: The mean object transfer time decreases despite increased load. The mean foreground object transfer time for overload factor 1.69 has a significantly larger confidence interval than for any other parameterization. This is an indication that the system behaved significantly differently over longer periods of time. We assume that there has been a synchronization between RADICCO's and the sender CCA's control loops.

For uTP-controlled background traffic, there are two about linear correlations recognizable: On the one hand, the higher the fraction of background subscribers, the higher the load that can be accepted without QoE impairment for the foreground traffic. And on the other hand, beyond the limit of this penalty-free overload, the increase in added transfer time also grows about linearly with the load. Remarkably, the mean transfer time even decreases a bit with increasing load for



(a) Applying RADICCO

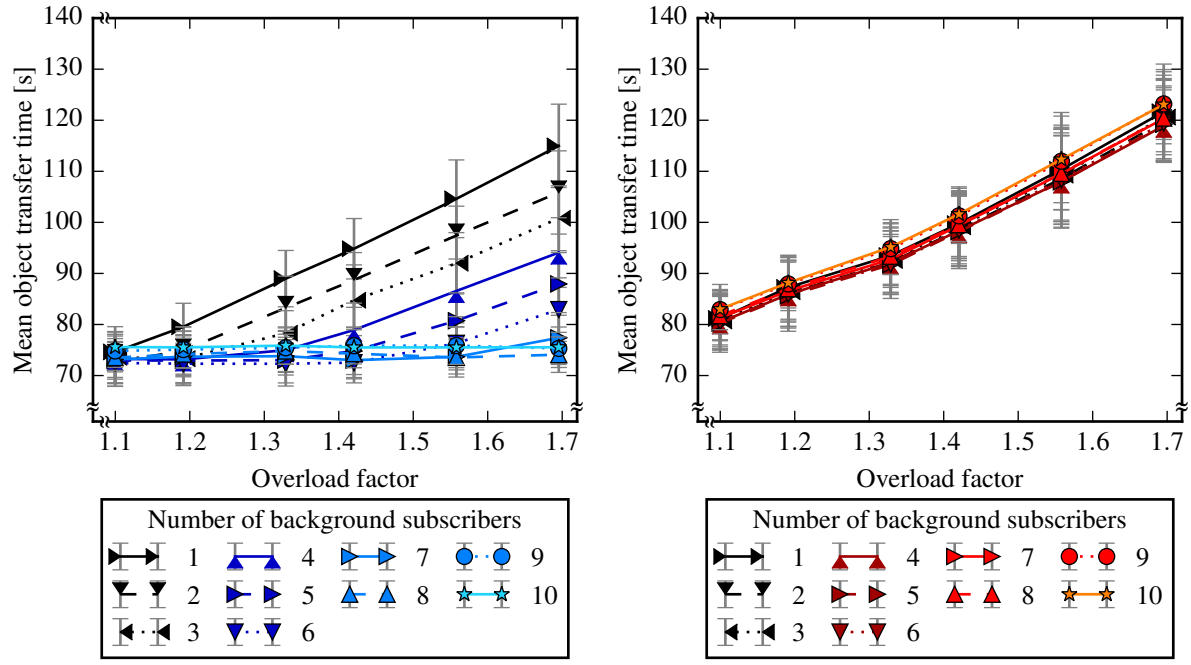
(b) Applying the reference HFS

Figure 4.6: Foreground object transfer times for the DEEP topology with clustered background subscriber distribution, the traffic model of software updates and TCP Vegas-controlled background traffic

high numbers of background subscribers, for ten background subscribers. As we will see in the next section, this advantage is paid for by a slightly lower utilization. We will explain the reasons for that effect when discussing utilization.

DEEP Topology

For the DEEP topology, Figure 4.6 shows the mean foreground transfer times in case of TCP Vegas-controlled background traffic, while Figure 4.7 shows the mean foreground transfer times in case of uTP-controlled background traffic. For this topology and subscriber distribution the reference HFS does not result in clean linear mean object transfer times as visible in Figures 4.6b and 4.7b. The reason is that for low overload, the random arrival pattern combined with the strict fair rate distribution of the scheduler result in times of high competition on the one hand, resulting in these rather high transfer times, and on the other hand in idle times, resulting in a rather low utilization as will be shown in Section 4.4.2. The results are very similar to the ones of the BROAD topology, that means that RADICCO achieves its goals for both resource reallocation levels, the AGS interface as well as the BNG interface. As for TCP Vegas-controlled background traffic, we see a significant irregularity for high numbers of background subscribers for TCP-Vegas-controlled background traffic, here for ten background subscribers at overload factor 1.59. We cannot identify the root cause. The confidence interval is significantly larger than for the other parameterizations, so this might be the result of synchronization during significant periods of time. Another remarkable aspect of the results for uTP-controlled background traffic is that a higher number of background subscribers in some cases results in an increase of mean



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.7: Foreground object transfer times for the DEEP topology with clustered background subscriber distribution, the traffic model of software updates and uTP-controlled background traffic

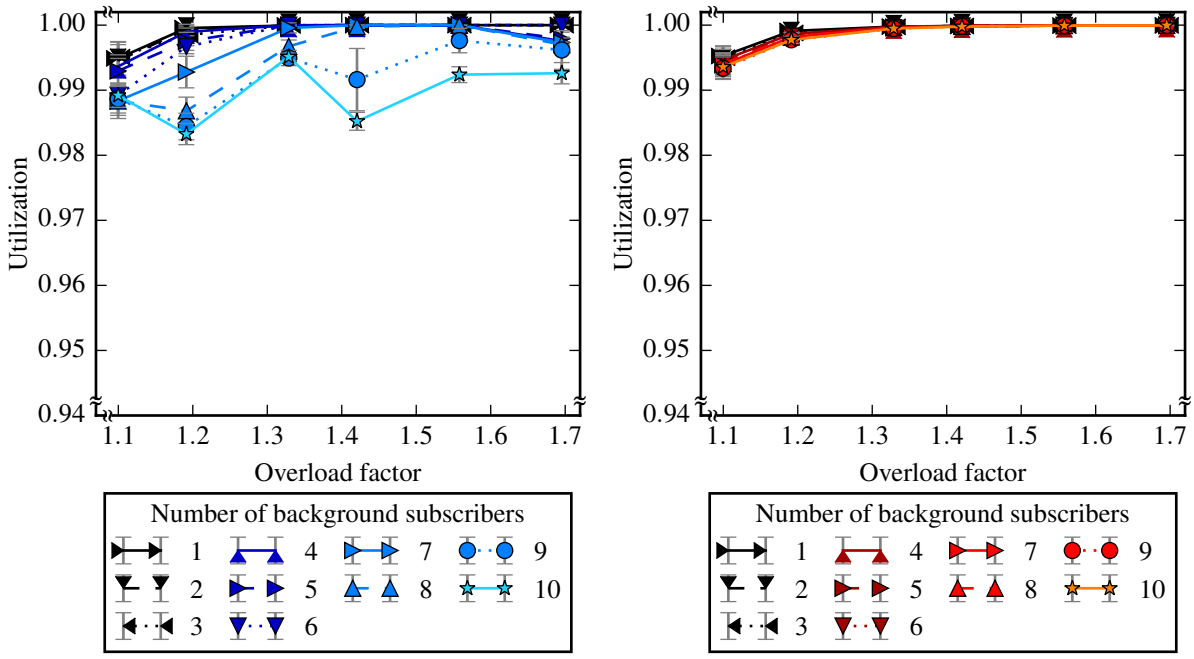
transfer time for foreground subscribers. In particular, nine or ten background subscribers result in slightly higher foreground transfer times compared to fewer background subscribers, which is also not the case for the BROAD topology. This may be caused by the senders' uTP CCAs entering hibernation mode less frequently than for the BROAD topology, and thus in average consuming more bandwidth. It is not clear what causes this change in behavior of the CCAs. However, the improvement regarding transfer times, the QoE-determining metric for the modeled service, is substantial also for these measurements, so the objective is met.

Summary

Summarizing, RADICCO results in significant benefit in terms of QoE for foreground transfers according to the software updates traffic model, regardless on which level the weight adaptation is necessary. This benefit is more clear if background traffic is controlled by uTP rather than if it is controlled by TCP Vegas.

4.4.2 Bottleneck Utilization

In this section, we analyze if the price for the foreground traffic's prioritization is higher than necessary, i.e. to which extent RADICCO allocates the remaining bandwidth to background traffic or leaves it unallocated, thus wasted. First, we evaluate the performance of RADICCO facing background traffic controlled by TCP Vegas for the BROAD topology. Figure 4.8 shows



(a) Applying RADICCO

(b) Applying the reference HFS

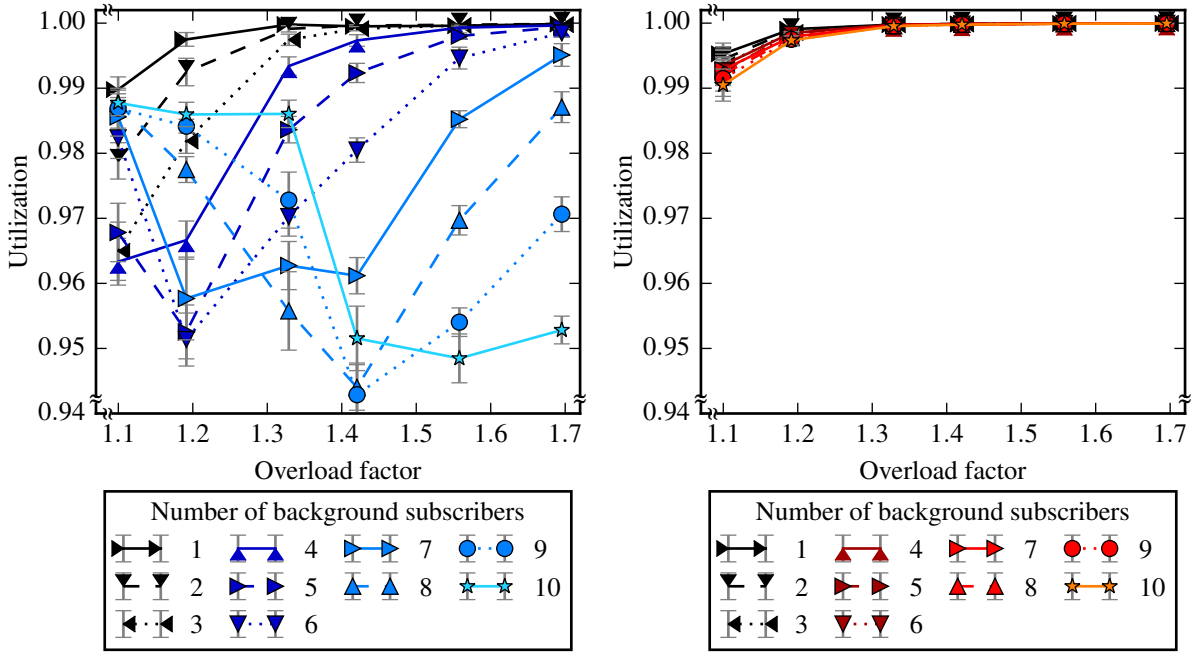
Figure 4.8: BNG interface utilization for the BROAD topology, the traffic model of software updates and TCP Vegas-controlled background traffic

the utilization of the BNG interface, the relevant bottleneck. Figure 4.8a shows the utilization when applying RADICCO, Figure 4.8b shows the utilization when applying the reference HFS. It can be clearly seen that applying RADICCO—and by this constantly changing weights and maximum rates—comes to a price of a slightly reduced bottleneck utilization. Nevertheless, the utilization is always above 98.5 % which is an acceptable value if there are other benefits achieved by deploying RADICCO.

Note that although on average all plotted scenarios offer higher load than the BNG interface can handle, due to the stochastic nature of the arrivals both schedulers do not achieve full utilization for overload factor 1.1.

There is significant decrease in utilization for overload factor 1.44 for nine or ten background subscribers. Interestingly, for ten background subscribers all confidence intervals are small. This indicates that the irregularity is not a consequence of too few measurements, but is a steady system state. Probably, the cause for these differing utilizations are synchronizations between the CCA of some senders and RADICCO. Due to the random arrivals and thus changing load, the synchronization cannot be stable as such. Yet, since the confidence intervals are small, the time share of synchronization is stable over the evaluated simulation batches. So probably, some constellations that occur frequently for ten background subscribers and less frequently for nine background subscribers (see the confidence interval) result in a synchronization that results in reduced utilization.
Nevertheless, the achieved utilization is satisfactory.

We show results for analog simulations, i.e. BROAD topology and software updates traffic model, using uTP-controlled background traffic in Figure 4.9. Figure 4.9a shows the load of



(a) Applying RADICCO

(b) Applying the reference HFS

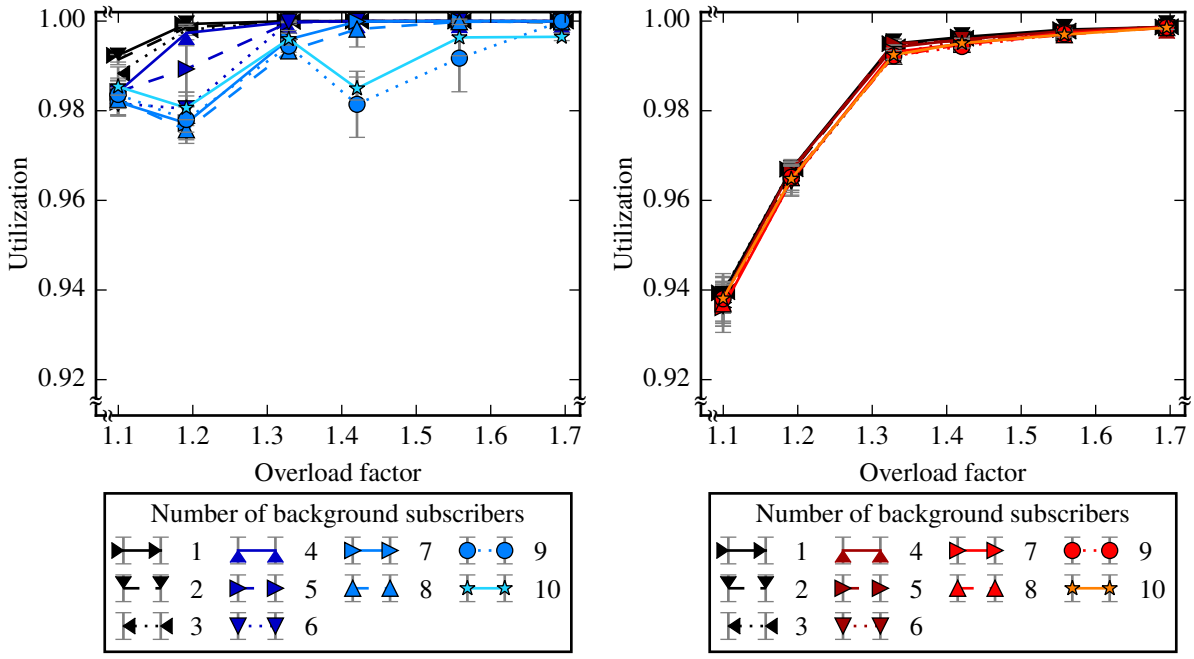
Figure 4.9: BNG interface utilization for the BROAD topology, the traffic model of software updates and uTP-controlled background traffic

the BNG interface when applying RADICCO, Figure 4.9b shows the load when applying the reference HFS. Again, the utilization is affected by applying RADICCO. Utilization is still on an acceptable level, at any time higher than 94 %, but significantly lower than in case of TCP Vegas. Again, we see no full utilization for low stochastic overloads for both schedulers, but already at a load of 1.21 the HFS is able to nearly fully utilize the link. This indicates that uTP can seize freed bandwidth faster than TCP Vegas in that scenario.

When examining high loads with high numbers of uTP-controlled background subscribers, we see a substantially lower utilization compared to HFS but also compared to RADICCO when managing TCP Vegas-controlled background traffic. Obviously, RADICCO's rate reduction scheme and uTP's congestion detection scheme tend to collide from time to time, resulting in background subscribers not utilizing their assigned bandwidth. Here the hibernation feature of uTP plays an important role. At the same time, this freed bandwidth (compared to lower overload) allows foreground subscribers to be served ahead of time and in consequence on average receiving slightly more than their calculated share. This explains the effect of even slightly decreasing mean transfer times with increasing load for high numbers of background subscribers.

Again, we see several load levels with a dip at an overload factor 1.44. The narrow confidence intervals indicate that these dips are not the result of single coincident completion times. Due to the black box CC implementations used in this evaluation, such effects evade detailed analysis.

We also evaluate the utilization for the DEEP topology with the clustered distribution of background subscribers. Figure 4.10 shows the utilization of the BNG interface, the relevant bottleneck. Figure 4.10a shows the utilization when applying RADICCO, Figure 4.10b shows the



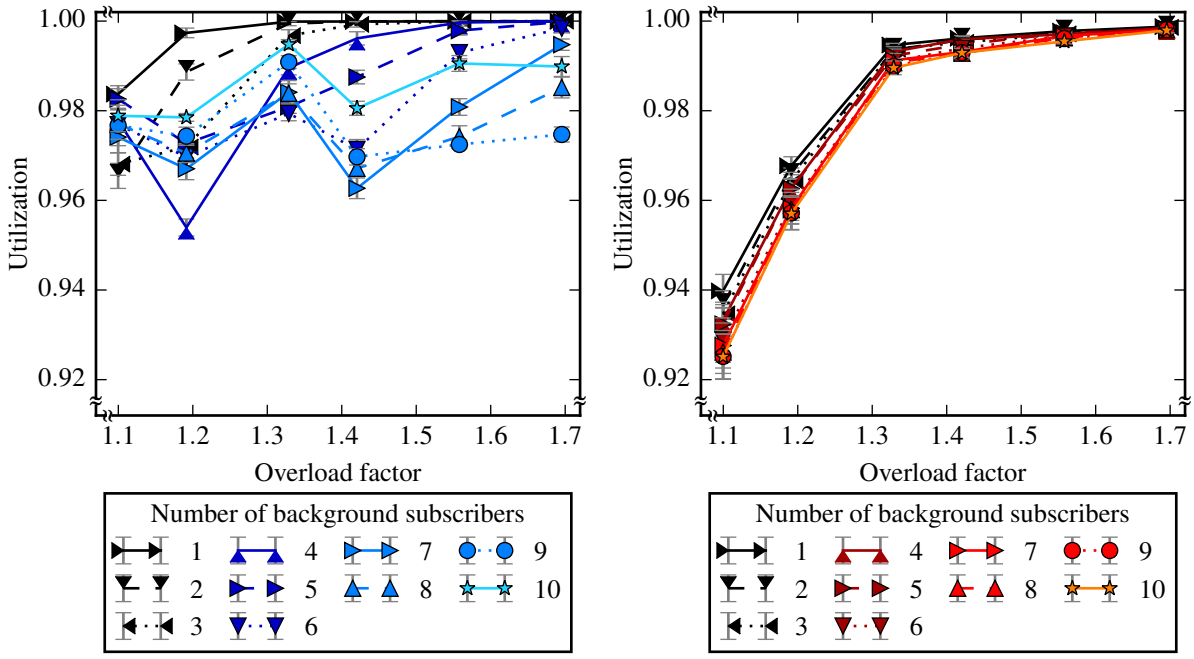
(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.10: BNG interface utilization for the DEEP topology with clustered background subscriber distribution, the traffic model of software updates and TCP Vegas-controlled background traffic

utilization when applying the reference HFS. For RADICCO, we see very similar values as for the BROAD topology, although there is a general shift to slightly smaller utilization values. Furthermore, there two interesting aspects in these results: First, RADICCO achieves a higher utilization at overload factors 1.1 and 1.21 than the reference HFS. This is caused by RADICCO reducing the rates of background subscribers and thus stretching their transfers in time. For the software updates traffic model with its delay-based definition, this also means that the fraction of time with active transfers increases. Therefore, these stretched background transfers form a more continuous load and thus reduce the impact of randomly overlapping durations of inactivity of different subscribers.
Second, the utilization for the simulations with nine and ten subscribers both show a similar pattern with lower utilization for overload factors 1.21 and 1.44. We therefore assume that in these cases synchronization between RADICCO's control loop and the control loops of the TCP Vegas senders occurs for some time. This synchronization might be amplified by the small number (five) of leaf nodes per edge node in the DEEP scenario.

The evaluation of utilization for the DEEP topology with the clustered distribution of background subscribers and uTP-controlled background traffic is shown in Figure 4.11. Figure 4.11a shows the results when applying RADICCO, Figure 4.11b the results for the reference HFS. As for TCP Vegas-controlled background traffic, RADICCO achieves a higher utilization for the same reasons. Again, the results for overload factor 1.21 and 1.44 are noticeable: For most traffic compositions, they result in significantly lower utilization than neighboring results. We consider the small number of leaf nodes per edge node being the cause for this effect. More interestingly, the comparison between the two topologies, i.e. of Figures 4.9a and 4.11a, shows that RADICCO



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.11: BNG interface utilization for the DEEP topology with clustered background subscriber distribution, the traffic model of software updates and uTP-controlled background traffic

achieves similar results for both topologies and even slightly better utilization for the DEEP topology.

4.4.3 Fairness among Foreground Subscribers

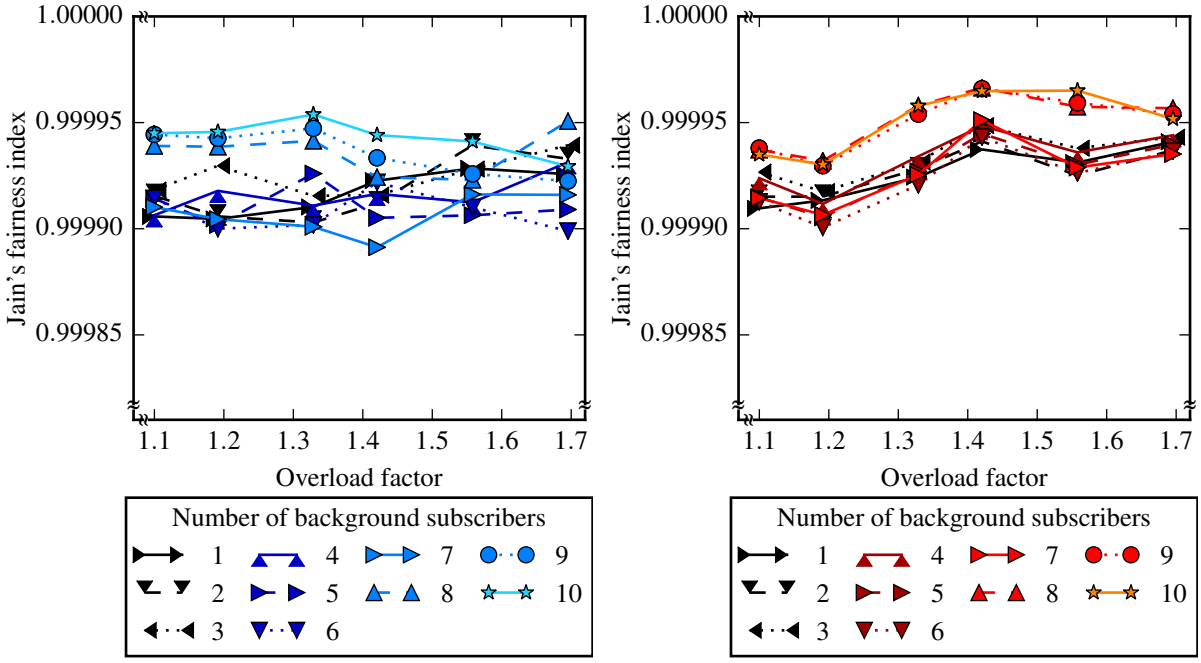
Figure 4.12 shows Jain's fairness index for the foreground subscribers for the BROAD topology and uTP-controlled background traffic, in detail Figure 4.12a for RADICCO, Figure 4.12b for the reference HFS. Note that there are no confidence intervals available for this metric.

The level of fairness is close to perfect fairness, all calculated fairness values are less than 0.2 ‰ from perfect fairness. Considering that these are results not of constant greedy transmissions but of transmissions of random objects, this level by far exceeds the required or desired fairness.

For TCP Vegas-controlled background traffic the figures look very similar and there are more interesting results for other metrics, so we refrain from including the TCP Vegas results.

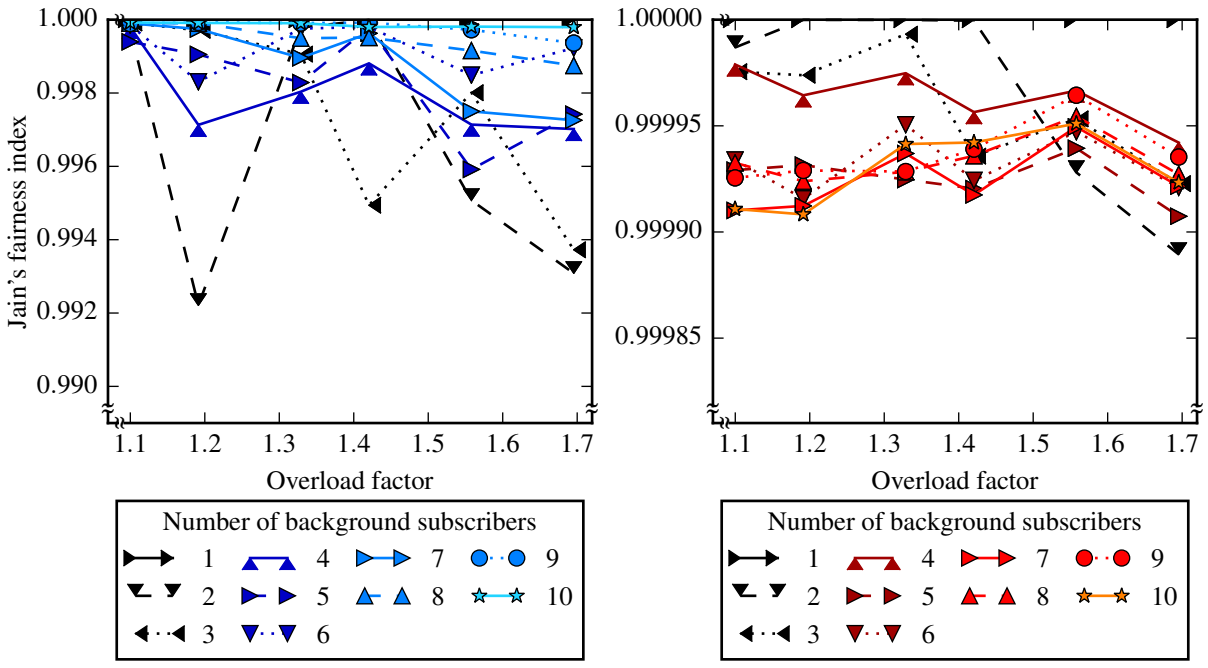
4.4.4 Fairness among Background Subscribers

Figure 4.13 shows Jain's fairness index among the background subscribers for the BROAD topology and uTP-controlled background traffic. Figure 4.13a on the left shows the results when applying RADICCO. There are two major observations regarding RADICCO's results: First,



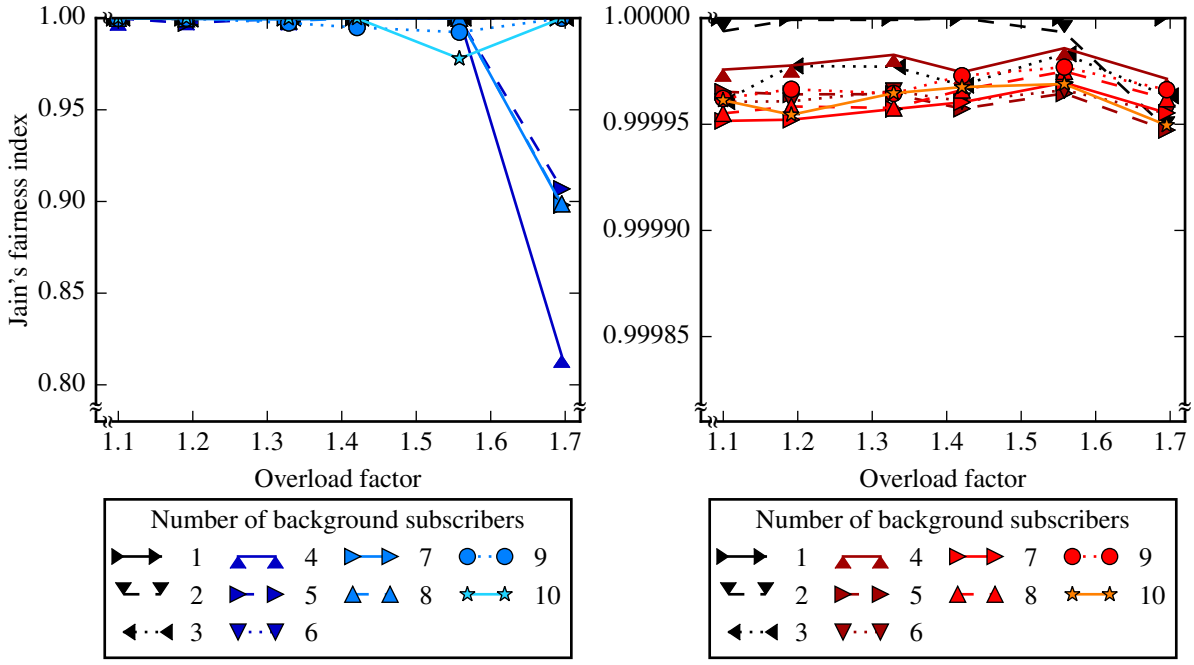
(a) Applying RADICCO (b) Applying the reference HFS

Figure 4.12: Jain's fairness index of the foreground traffic for the BROAD topology, the traffic model of software updates and uTP-controlled background traffic



(a) Applying RADICCO (b) Applying the reference HFS

Figure 4.13: Jain's fairness index of the background traffic for the BROAD topology, the traffic model of software updates and uTP-controlled background traffic



(a) Applying RADICCO

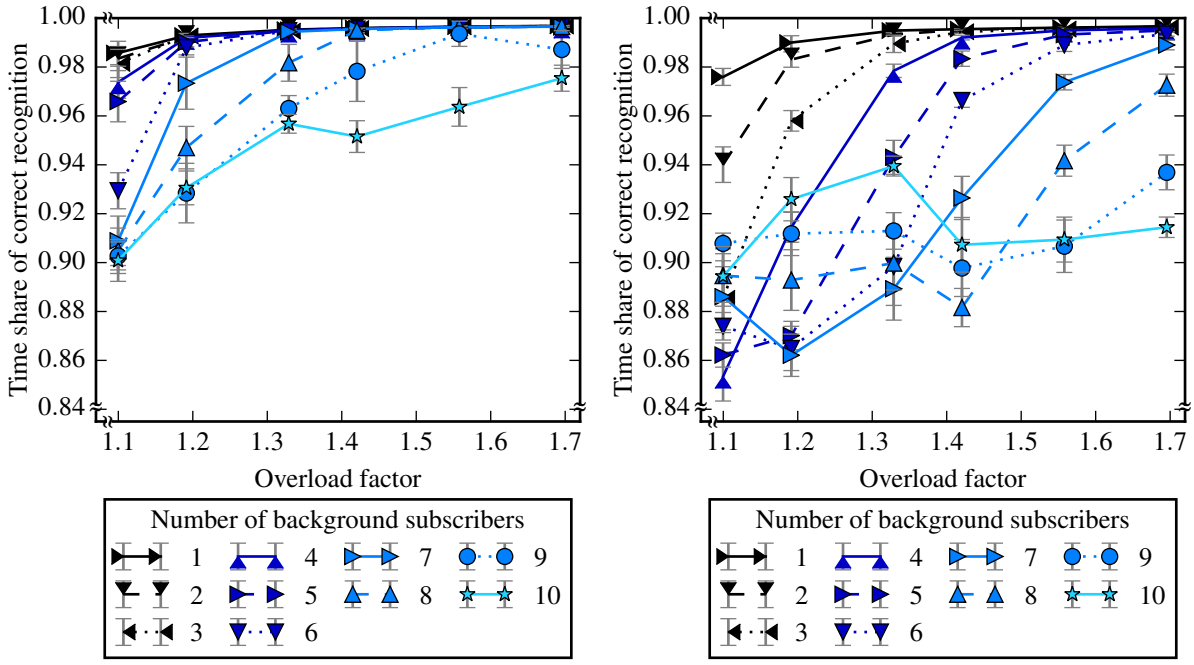
(b) Applying the reference HFS

Figure 4.14: Jain's fairness index of the background traffic for the BROAD topology, the traffic model of software updates and TCP Vegas-controlled background traffic

the range of fairness is much broader, consider the different scale of Figure 4.13a compared to Figures 4.12b, 4.12a and 4.12b. Second, there is more volatility and less a clear order among the different lines. This is probably caused by the increased dynamics introduced by RADICCO. Assessing these results, the performance is still more than acceptable, Jain's fairness is less than 1 % off the perfect value for all experiments.

As expected, the reference scheduler does not show a significant difference between foreground and background regarding fairness, as also the comparison of Figures 4.12b and 4.13b shows (same scale).

The evaluation of RADICCO's fairness when facing TCP Vegas-controlled background traffic shows surprising differences. Figure 4.14 shows Jain's fairness index for the background subscribers' shares for the BROAD topology and TCP Vegas-controlled background traffic, Figure 4.14a shows results for application of RADICCO, Figure 4.14b the results for the reference scheduler. Again, there are no unexpected effects for the reference scheduler, the achieved fairness is close to perfect fairness. Yet the result for RADICCO are by far not as good as in case of uTP-controlled traffic. While some subscribers receive bandwidth close to that of the foreground subscribers, others receive very little bandwidth only. For example, a detailed analysis of the lowest fairness index for four background subscribers and a configured stochastic load of 1.69 shows that there are three subscribers that achieve an average throughput of 10.48–10.58 Mbit/s while the fourth receives an average throughput of 1.46 Mbit/s. This effect is probably caused by a negatively reinforcing synchronization between the control loops of RADICCO and the TCP Vegas CCA. Yet, this is a rare case as we see from the overall results for TCP Vegas-controlled background traffic. Nevertheless, this last figure alone is not considered



(a) For TCP Vegas-controlled background traffic

(b) For uTP-controlled background traffic

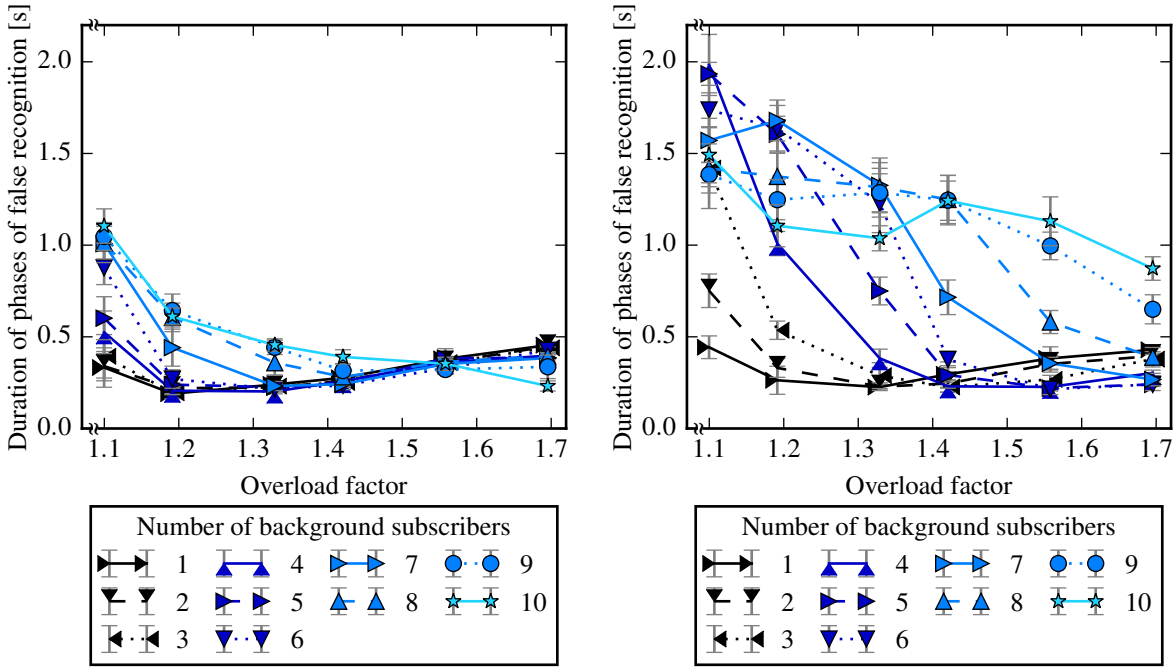
Figure 4.15: Time share of correct recognition of foreground traffic for the BROAD topology and software updates

too low. When applying RADICCO, we explicitly aim at reducing the bandwidth background subscribers receive under such heavy overload. For instance, we are satisfied with the results of RADICCO handling uTP background subscribers, but in the same scenario it means that all four background subscribers receive on average 20.0 – 22.7 kbit/s only.

So, these results, along with the object transfer times for this scenario (shown in Figure 4.4a), indicate that RADICCO does not function well in some cases when facing TCP Vegas traffic. We will provide conclusions in the conclusion for the software updates scenario in Section 4.4.7. Indeed, the chosen parameterization of RADICCO in combination with these scenarios causes RADICCO to often recognize TCP Vegas-controlled background traffic as foreground traffic, as the analysis of the foreground recognition durations in the next Section shows.

4.4.5 Correct Recognition of Foreground Traffic

Figure 4.15 shows the time share of correct foreground identification of RADICCO, in Figure 4.15a for TCP Vegas-controlled, in Figure 4.15b for uTP-controlled background traffic. Interestingly, although we only show the recognition of foreground subscribers using TCP Cubic, it makes a significant difference whether TCP Vegas or uTP is used by the background subscribers. In both cases, we see non-negligible time shares of false recognition. Therefore, we also analyze the duration of phases of false recognition. In Figure 4.16 we show the duration of the phases of false recognition of foreground subscribers, in Figure 4.16a for TCP Vegas-controlled, in Figure 4.16b for uTP-controlled background traffic. Clearly, the phases of false recognition are smaller when competing with TCP Vegas-controlled background traffic. Nevertheless, uTP's



(a) For TCP Vegas-controlled background traffic

(b) For uTP-controlled background traffic

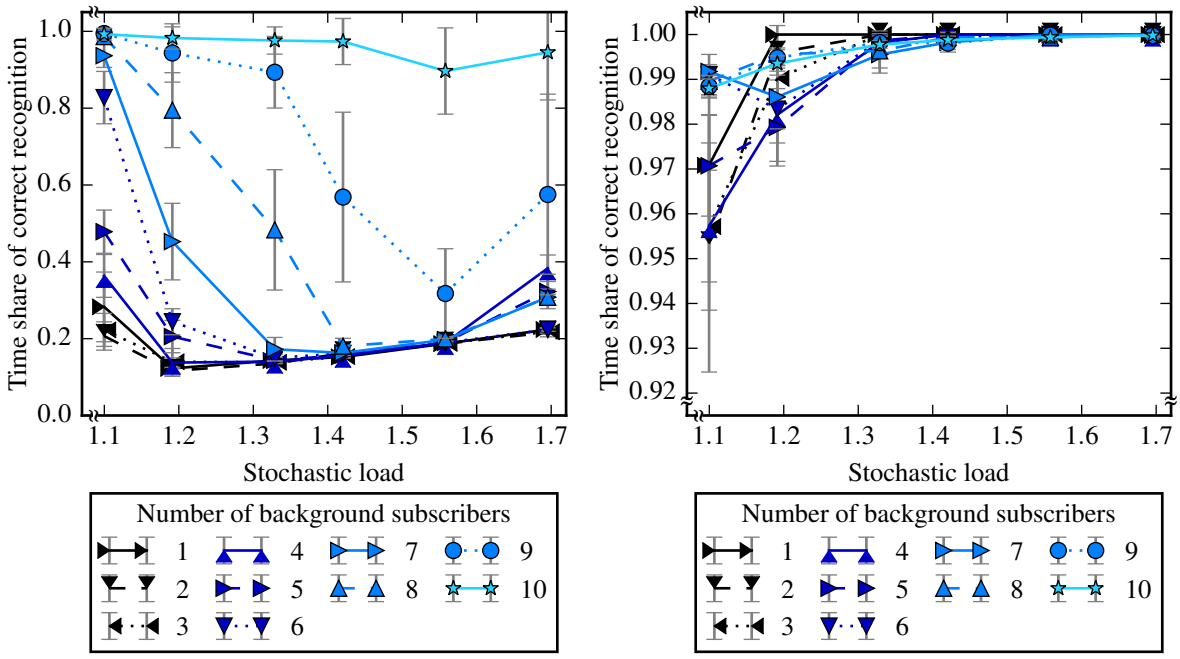
Figure 4.16: Duration of phases of false recognition of foreground traffic for the BROAD topology and software updates

very defensive CCA leaves some bandwidth unused as we saw in Figure 4.9. Moreover, as we will show next, uTP-controlled traffic is much more reliably recognized as background traffic by RADICCO's recognition algorithm. All in all, despite longer phases of false recognition of foreground traffic uTP-controlled background traffic even results in lower transfer times for foreground traffic than TCP Vegas-controlled background traffic as we showed in Figures 4.5a and 4.4a.

Therefore, we conclude that the detection of foreground subscribers works well when competing with any of the two background CC.

4.4.6 Correct Recognition of Background Traffic

In Figure 4.17 we show the time share of correct recognition of background traffic, in Figure 4.17a of TCP Vegas-controlled and in Figure 4.17b of uTP-controlled background traffic. Clearly, TCP Vegas traffic is not correctly recognized most of the time. This is probably caused mostly by the unsteady load: The more slowly-reacting and therefore load-stabilizing TCP Vegas flows exist, the better the background recognition works. So, a scenario with higher BNG interface capacity and more active subscribers probably would result in significantly more correct background recognition also for TCP Vegas traffic. A potential remedy for this behavior is to use a substantially higher buffer utilization threshold for foreground detection but there are substantial drawbacks, see the discussion in Section 3.8.5.
These results on background recognition also help to explain that the foreground recognition works better for TCP Vegas-controlled background traffic than for uTP-controlled background



(a) For TCP Vegas-controlled background traffic

(b) For uTP-controlled background traffic

Figure 4.17: Time share of correct recognition of background traffic for the BROAD topology and software updates

traffic: Since TCP-Vegas subscribers are recognized as foreground subscribers most of time, RADICCO executes less rate adaptations, thus the rate of each foreground subscriber is more stable. Yet, we know for TCP Cubic, the foreground CCA, to never reduce the queue below the used relative buffer usage detection threshold $tt\_recog^{relQThresh} = 0.35$ in steady state. So, the bad recognition performance for TCP Vegas makes the system steadier from the foreground subscribers' CCAs, which in turn avoids low buffer utilization of foreground subscribers. This corresponds to reducing the false recognition durations for foreground subscribers.

Concluding, background recognition works well for uTP-controlled traffic, but not as well for TCP Vegas-controlled traffic. This also limits the achieved QoE improvement, as shown in Section 4.4.1, so an improvement of the recognition is desirable. The results indicate that RADICCO's reactions to rapid changes in load are the root cause for the unreliable background recognition. Therefore, RADICCO's recognition of TCP-Vegas-controlled background traffic is expected to work substantially better for ANs that aggregate more access links, since in that case the overall load is smoother.

For space reasons, we will not show results on traffic recognition for other scenarios.

4.4.7 Scenario Conclusion

The evaluation of the update distribution scenario shows that RADICCO basically works. We show that the crucial metric for foreground software update downloads, the object transfer times, is significantly improved when using RADICCO compared to using the reference HFS.

Moreover, we show that other important metrics, bottleneck utilization, and fairness among foreground subscribers, are not significantly impaired. In contrast, the fairness among background subscribers is far less than perfect, although we do not regard this metric being important for the performance evaluation of RADICCO. It merely indicates a potential field of research when aiming for improvement. We also show that the recognition of subscribers works very well for foreground traffic and uTP-controlled background traffic. For TCP Vegas-controlled background traffic though, the recognition does not reliably work correctly. Yet, background recognition for TCP Vegas is expected to work substantially better for ANs that aggregate more access links. Moreover, in contrast to uTP, TCP Vegas is not in use as background CCA in the Internet.

Summarizing, RADICCO achieves a significant improvement for the QoE-relevant metric for the traffic type of software updates.

4.5 Performance for Video on Demand Streaming Traffic

This scenario focuses on the impact of RADICCO on subscribers receiving a VoD streaming service while other subscribers served by the same BNG downstream interface receive software updates using a background transport protocol. So, the most important measurement are the transfer times of the video model objects which represent the single chunks of data transmitted in a DASH-like VoD service. In every simulation, there is a group of ten subscribers receiving traffic according to the respective VoD model, and a second group of ten subscribers receiving traffic according to the software updates model. Among the second group, the number of subscribers using a background CC is varied from one to ten subscribers. We use the software updates traffic rather than greedy traffic to show that RADICCO achieves benefits also if background traffic is non-continuous and therefore jumps in load and assigned rates occur frequently.

We performed simulations for the two traffic models for DASH-like VoD streaming services for both the BROAD and the DEEP topologies. The results for the DEEP topology confirm that RADICCO also works for multi-level topologies but provide no further insights, so we will show only results for the BROAD topology. We will first look into the transfer times of the objects of VoD traffic. Second, we will also examine the bottleneck utilization.

4.5.1 Transfer Times of Foreground Traffic

In this scenario and for this metric, we pursue a second, fundamentally different evaluation approach compared to the other scenarios: Here we use simulation also to assess the limits of stable operation, not only to evaluate performance in steady state configurations. These limits exist since the traffic in this scenario consists not only of elastic traffic, the updates, but also of non-elastic traffic, the VoD traffic.

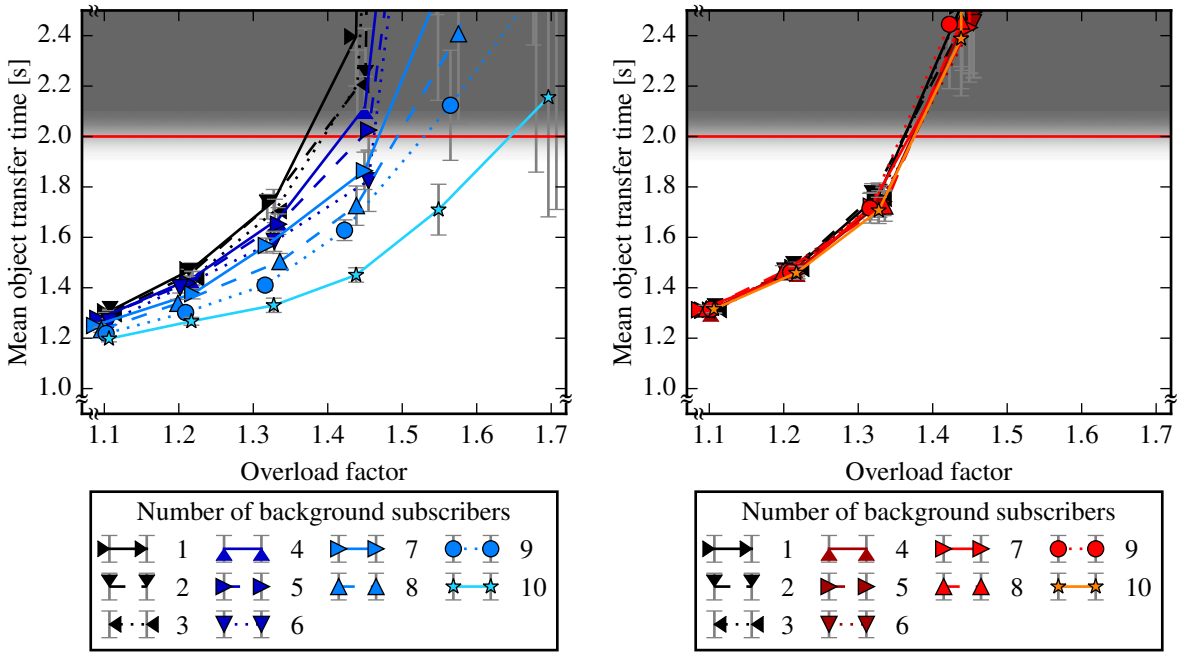
Since we model VoD traffic as non-elastic with IATs, in theory there exist two areas of system parameterization:
First, a stable area, in which the scheduler can allocate sufficient resources for VoD traffic. This means that on average the last object's transmission is finished when its successor's transmission begins. This does not mean that all transfer times consist of transmission time only, there may

exist a waiting time needed for the predecessor transmission to complete. Nevertheless, on average the sum of both is smaller or equal to the expectation value of the IAT. More precisely, the expectation value of the transfer time of an object is smaller or equal than the expectation value of the IAT, which is determined by the known traffic model. Nevertheless, the expectation value of the transfer time for this dynamic system with the traffic controlled by complex CCAs is not known and we are only able to simulate a limited number of transmissions to estimate the expectation value. Thus, the calculated estimate comes with some uncertainty.
Second, the area of saturation, in which the application layer system is overloaded, i.e. transfer demands queue up. So, the expectation value of the transmission time (i.e. without waiting time) is greater than the expectation value of the IAT. This immediately results in the expectation value for the transfer time (including waiting time) being infinite.

With our simulations, we face two issues here: First, we cannot measure infinite transfer times for saturated systems due to our finite simulation time. Instead, we will measure high, and mostly increasing transfer times for such systems. The calculated averages do not represent estimated expectation values in that case since these simulations violate the fundamental assumption of simulating a steady state system. Second, since we only simulate a finite number of transmissions in a complex system, calculated average transfer times come with some uncertainty. Therefore, the probability of false recognition (stable / saturated) is rather high for calculated average transfer times near the expectation value of the IAT. This obviously applies to all measurements with calculated confidence intervals ranging from below the threshold to above the threshold. Note that these intervals shrink with increasing simulation time but it neither was feasible nor worth the effort—the areas of uncertainty do not vanish—to run simulations longer. Because of similar reasons we also did not use a finer parameterization which would have allowed to better estimate the stable area of our artificial traffic model. We rather aim to generally show that RADICCO results in sufficient resources being allocated for this arbitrary VoD traffic model in more states of operation than by the reference scheduler. At any such point of operation applying RADICCO results in better QoE for the users consuming the VoD service than if applying the reference HFS.
In Figures 4.18 and 4.19 we indicate these aspects as follows.

- A red line indicates the expectation value of the IAT.
- A gradient gray background ranging from about some interval (100 ms for the short interval VoD traffic model) below to the same distance above this expectation value. This area is the area of uncertainty. This gradient indicates the probability that a calculated average in that range indicates a steady system state.
- A gray background above that area indicates that if the calculated arithmetic mean is in that area the used system parameterization leads to an unstable system on application level with respect to the non-elastic VoD traffic.

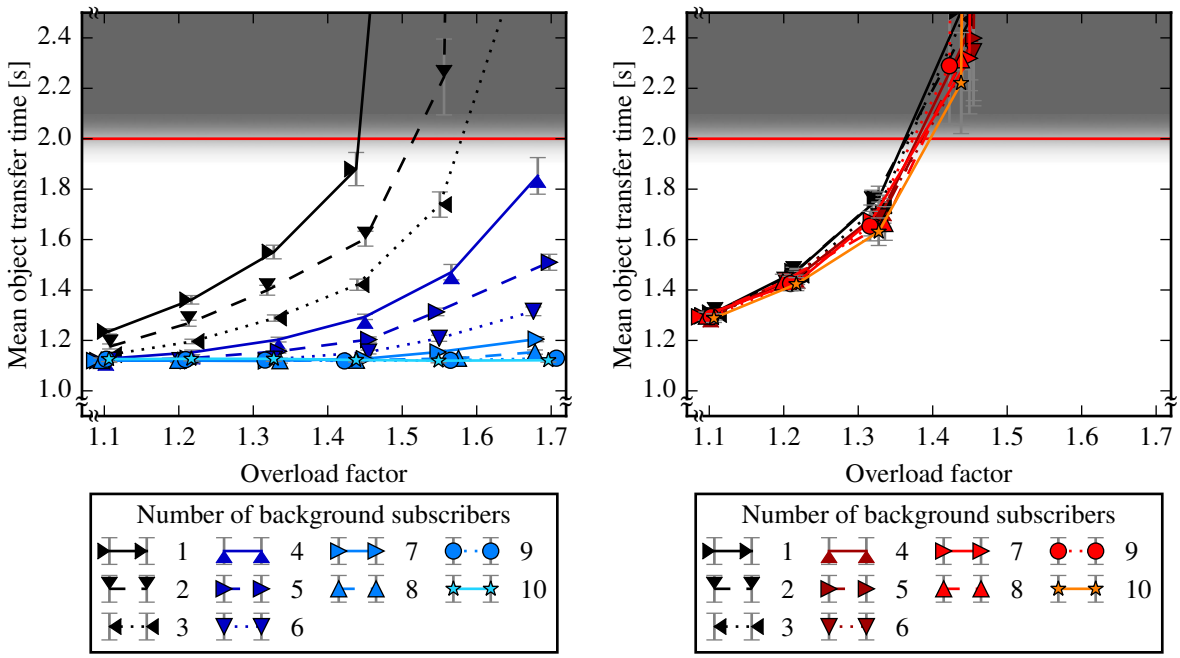
Figure 4.18 shows the mean transfer time of the short interval VoD traffic model when competing with TCP Vegas-controlled background traffic. Figure 4.18a on the left side shows the results when applying RADICCO, Figure 4.18b on the right side shows the results when applying the reference HFS. It is obvious that applying RADICCO reduces the object transfer times and, moreover, extends the range of network states allowing to transport this traffic.



(a) Applying RADICCO

(b) Applying the reference HFS

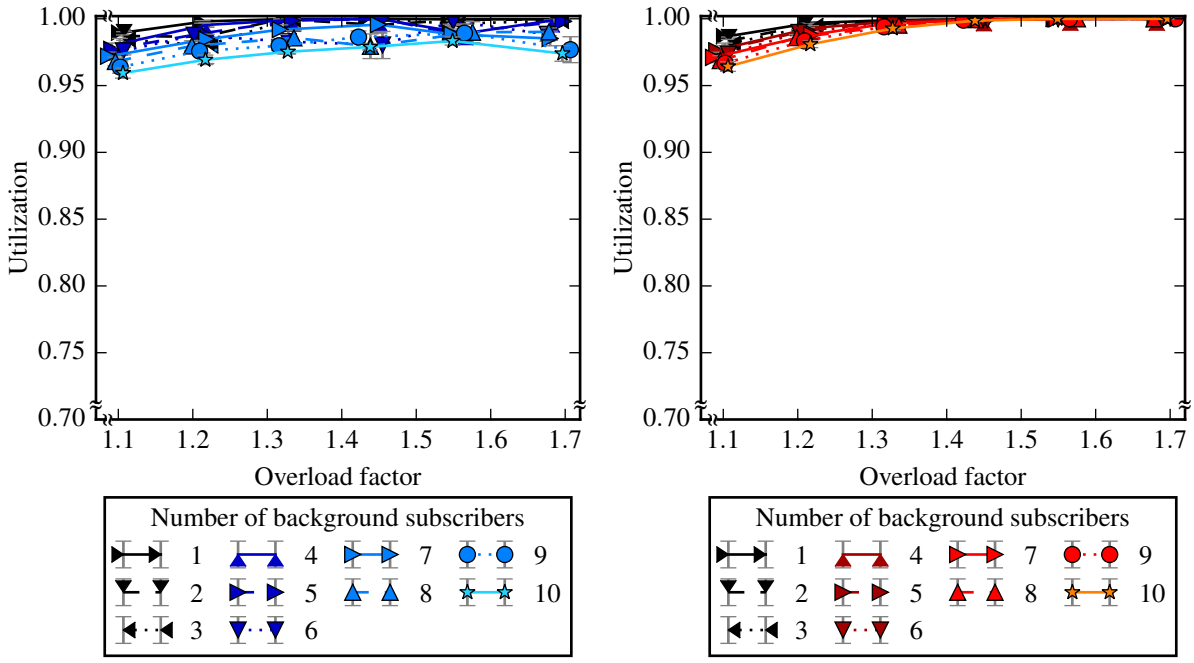
Figure 4.18: Foreground object transfer times for the BROAD topology, short interval VoD traffic model and TCP Vegas-controlled background traffic



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.19: Foreground object transfer times for the BROAD topology, short interval VoD traffic model and uTP-controlled background traffic



(a) Applying RADICCO

(b) Applying the reference HFS

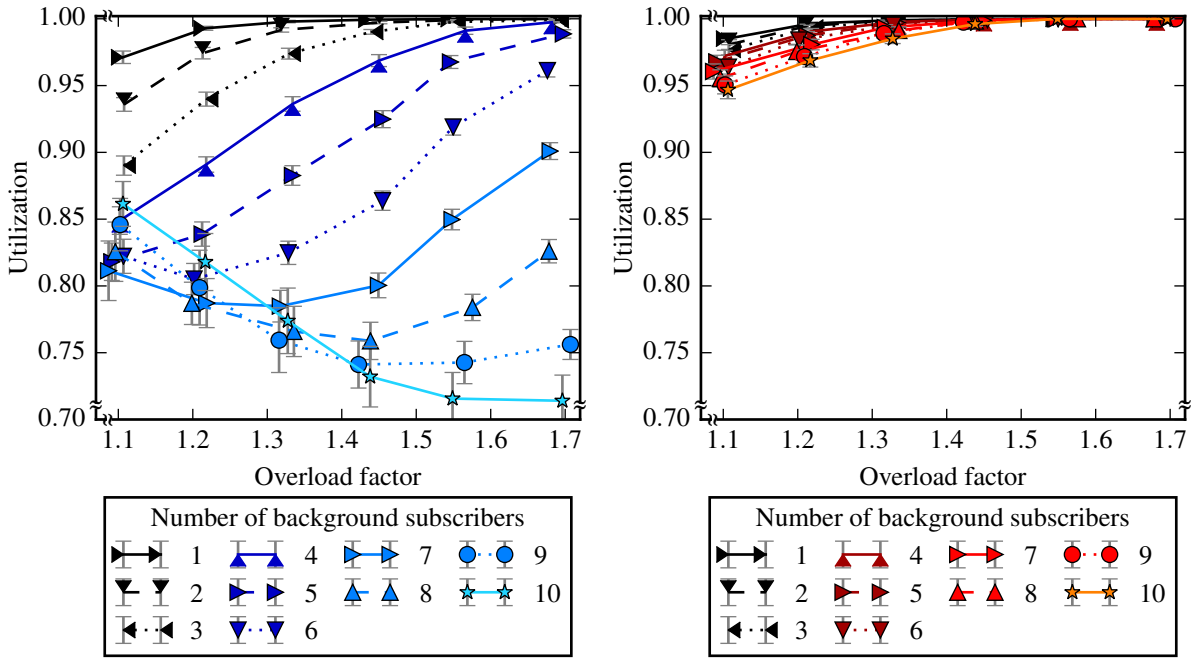
Figure 4.20: Utilization for the BROAD topology, short interval VoD traffic model and TCP Vegas-controlled background traffic

The reference HFS is only able to successfully transport the non-elastic VoD demand up to a load offer of 1.32 as Figure 4.18b shows. So, in these cases the VoD traffic is transported despite higher load offered than can possibly be carried. This is only possible since the elastic traffic share (the updates with their relative start time definition) receives much less bandwidth than it could make use of if the access link would be the only bottleneck. Thus, the VoD receives a significantly higher share in throughput than it has in offered load.
With RADICCO, the mean transfer times are substantially lower than when applying the reference HFS for most traffic compositions for these overload factors. As for the software updates traffic, we see that the higher the share of background traffic, the higher the gain. These reduced transfer times indicate headroom for even higher bitrate VoD transmissions at these points of operation. Nevertheless, this headroom cannot be quantified with our simulation approach.

Moreover, we see that several traffic compositions even allow safely transporting the VoD traffic for a load offer of 1.44, for which the reference HFS does not allow the VoD traffic to be transported for any traffic composition. The measurement for five background subscribers lies in the middle of the area of uncertainty. But for six and more background subscribers the means are well below the expectation value of the IAT of two seconds, so the VoD traffic is safely transported.

4.5.2 Bottleneck Utilization

The only other metric we show for this scenario is the bottleneck utilization. Since it is a packet level metric, there are no unstable systems. Figure 4.20 shows the utilization for the short interval



(a) Applying RADICCO

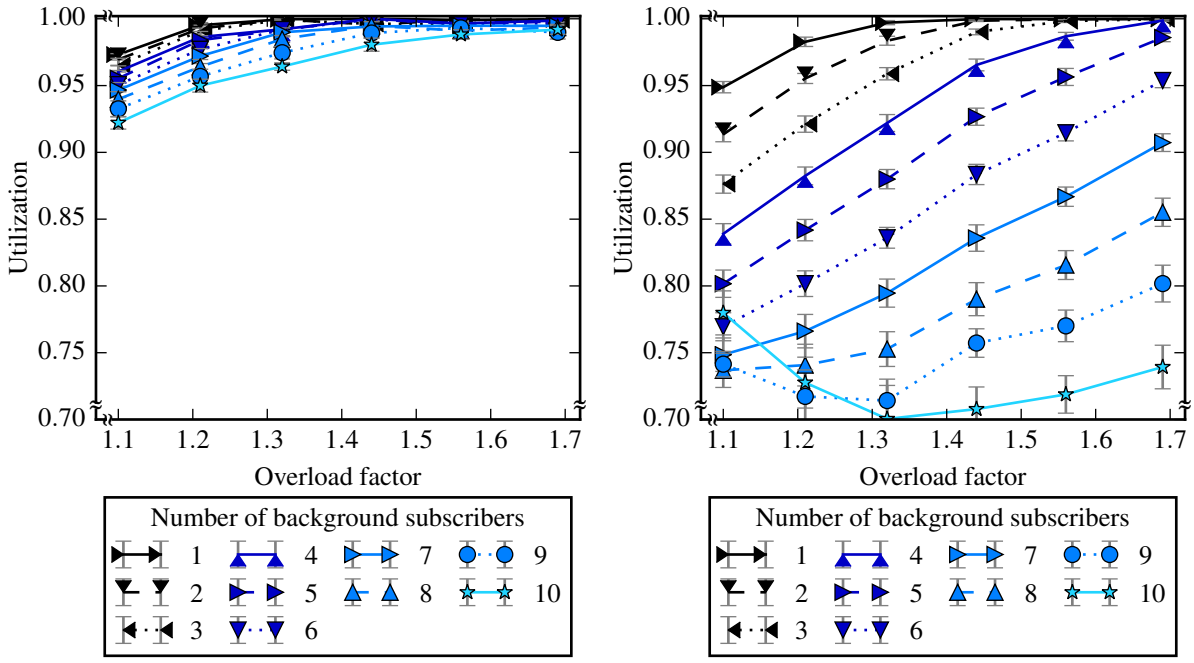
(b) Applying the reference HFS

Figure 4.21: Utilization for the BROAD topology, short interval VoD traffic model and uTP-controlled background traffic

VoD traffic model and background traffic controlled by TCP Vegas, Figure 4.20a when applying RADICCO, Figure 4.20b when applying the reference HFS. It shows that there is some price to be paid for the benefit in low transfer times and extended area of operation states allowing stable VoD delivery, nevertheless this price is rather low since utilization is well above 95 % for all measurements. Note that in order to ease comparison we deliberately show the same range in all figures on utilization in this section (Figures 4.20, 4.21 and 4.22).

The results for uTP-controlled background traffic are shown in Figure 4.21. Again, we show the results for RADICCO in Figure 4.21a and for the reference HFS in Figure 4.21b. Obviously, in case of uTP-controlled background traffic there is a rather high price to pay for the admittedly much bigger benefit that is achieved for the foreground VoD traffic when RADICCO adapts the scheduler's weights. The extent of underutilization is much bigger than for the software updates traffic shown in Figure 4.9a. It seems, that VoD streaming traffic results in a load so unsteady that the uTP CCA, known to yield generously to foreground traffic, falls back to a very low cwnd or even its hibernation mode. We see in the logs that the allocated bandwidth varies substantially, and frequently includes about one second without any packet being sent. Nevertheless, we also see in these throughput logs that the first probe is always successful, i.e. after such a first probe the throughput increases to several Mbit/s. This makes sense since the probe will find an empty queue at the scheduler and therefore detect no significantly increased OWD.

We already saw that this behavior did play a prominent role for the steadier software updates traffic model. It is of interest, whether the large interval VoD model results in sufficiently more steady load so that uTP's reaction is less extreme. The utilization achieved by the reference HFS for the large interval VoD traffic is like the one achieved for the short interval VoD traffic



(a) For TCP Vegas-controlled background traffic

(b) For uTP-controlled background traffic

Figure 4.22: Utilization for the BROAD topology, large interval VoD traffic model and RADICCO

model so we omit the respective figures. Figure 4.22 shows the utilization for the large interval VoD traffic when applying RADICCO. The results for TCP Vegas-controlled background traffic, shown in Figure 4.22a, do not surprise. Regarding uTP-controlled background traffic, shown in Figure 4.22b, clearly also this traffic is too unsteady to allow uTP to maintain a non-zero cwnd for longer times. Therefore, the achieved utilization is even lower than for the short interval VoD traffic model.

The results for VoD traffic competing with uTP-controlled background traffic deserve several remarks. First, we deliberately model a rather bad case for RADICCO with few subscribers served by one BNG downstream interface. In real networks, most topologies result in many more subscribers served by such an interface, see Section 2.1.1 and 2.1.2. Second, in this scenario our optimistic choice of uTP's target delay of 3 milliseconds may have a negative impact. In other scenarios, such low target delay pays off, but for highly volatile traffic such as VoD streaming a simulation study with higher targets is desirable. Yet, we did not carry out such study. Third and maybe most important: Although the bottleneck bandwidth is the scarce resource, QoE is still the primary goal. In these scenarios with uTP-controlled background traffic for small as well as large interval VoD, RADICCO achieves significantly lower transfer times for VoD than the reference HFS, although leaving up to 30 % of the bottlenecks capacity unused. Moreover, today's broadband access networks suffer from peak or overload just for a short period per day (see Section 2.1.3). So probably in today's access network context even such performance is more desirable than the performance of the reference HFS.

4.5.3 Scenario conclusion

Summarizing, VoD is a challenging traffic type for RADICCO, in particular in combination with uTP-controlled background traffic. RADICCO substantially extends the area of operation states that allows to successfully transport a certain DASH-like VoD traffic model. Moreover, it significantly reduces the object transfer times for lower overload, i.e. it increases the reliability of a maximum QoE service delivery and extends the headroom for more demanding VoD traffic. A drawback is that for uTP-controlled background traffic the application of RADICCO results in low utilization of the bottleneck in this scenario.

4.6 Performance for Web Browsing Traffic

In this scenario, we evaluate the effect of RADICCO on transfers of small objects. For these transfers, typically the transfer delay is decisive for QoE.

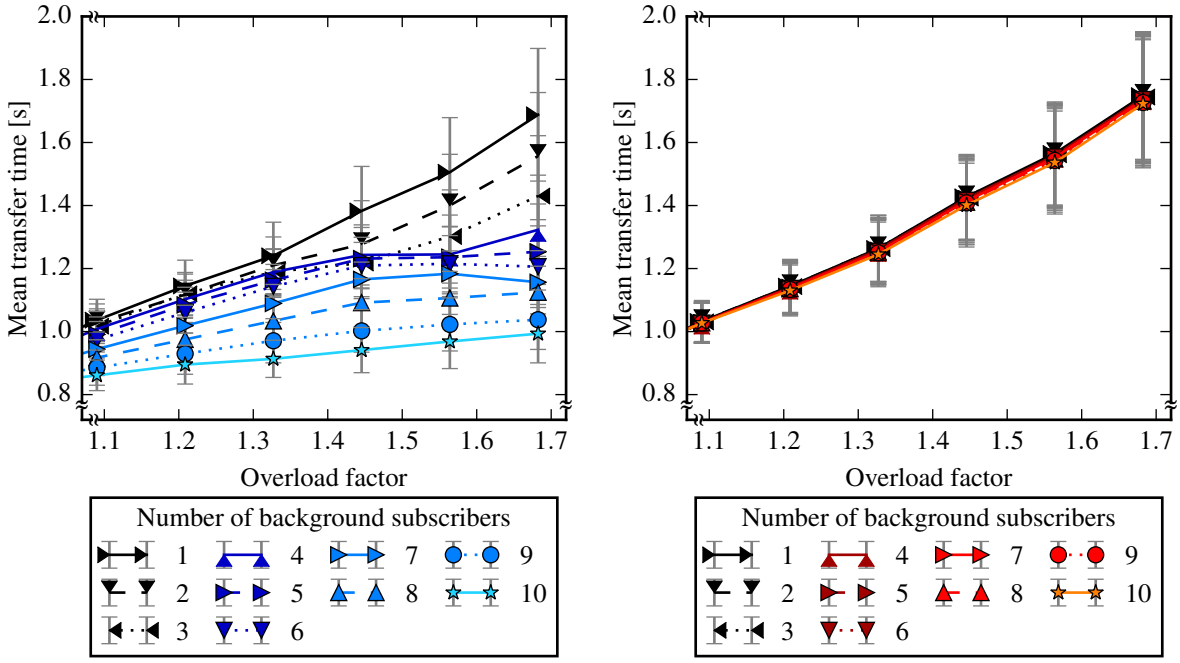
Since web browsing traffic contains large idle durations, the overall offered load by one web subscriber is very low. We therefore scaled the topology to ninety subscribers receiving traffic according to our web traffic model competing with ten subscribers receiving traffic according to our software updates traffic model. Again, among these, the number of background subscribers is varied.
Note that even though with this increased number of subscribers the offered load of web browsing accumulates to 27.2 Mbit/s only, while the offered load by the update downloads is 183.3 Mbit/s. This low share in offered load of about 15 % is a desired configuration since we aim to capture the sporadic nature of load caused by such traffic.

4.6.1 Transfer Times of Foreground Traffic

In the following, we show both absolute and relative results, i.e. absolute measurements for both RADICCO and the reference HFS as well as evaluate the relative change introduced by replacing the reference HFS by RADICCO.

Figure 4.23 shows the results for TCP Vegas-controlled background traffic, Figure 4.23a when applying RADICCO, Figure 4.23b when applying the reference HFS. Although less clearly, we again can recognize the double dependency already seen for the updates traffic model evaluated in Section 4.4: The more background traffic is available, the more overload can even be tolerated without significant impact and the more RADICCO can reduce the transfer times during overload.

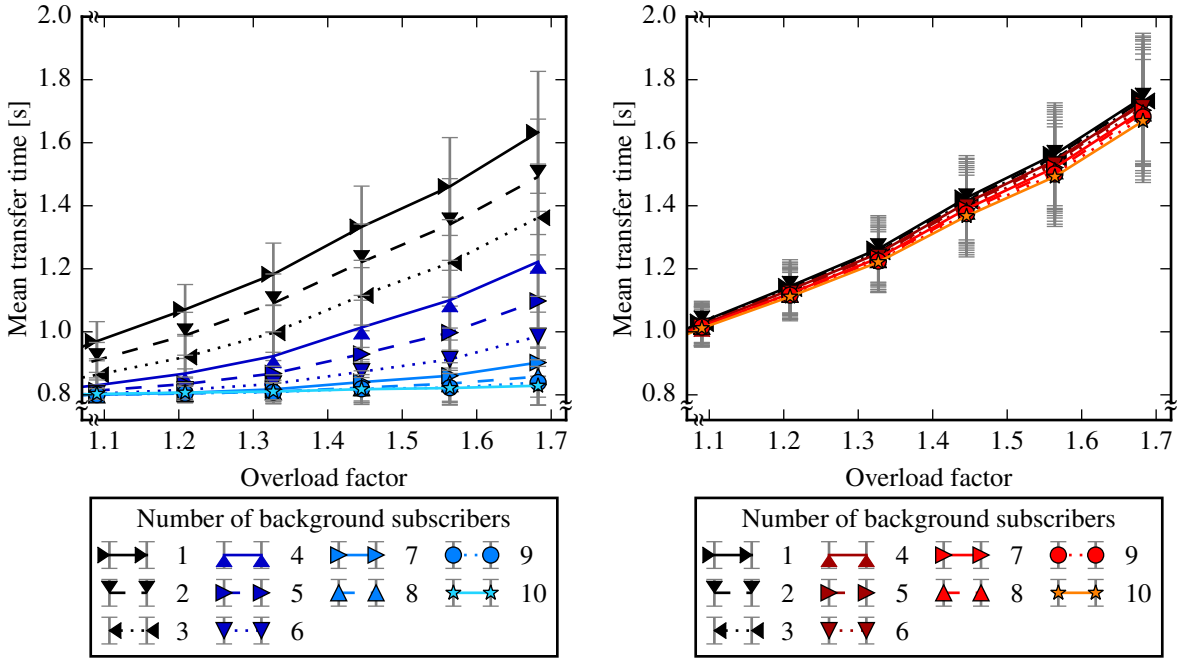
This effect is more distinct for uTP-controlled background, here the dependencies are again approximately linear. Figure 4.24 shows the respective results, Figure 4.24a when applying RADICCO, Figure 4.24b when applying the reference HFS. Here, RADICCO achieves a lower transfer time than the reference HFS for all measurements. Unfortunately, the broad unbalanced object size distribution of web traffic results in rather big confidence intervals even though the presented figures base on simulations we ran five times as long (10.000 s) as the other simulations. Because of the large confidence intervals, it appears that there is statistical uncertainty if there is a benefit of applying RADICCO if there are only few (1–3) background subscriber active.



(a) Applying RADICCO

(b) Applying the reference HFS

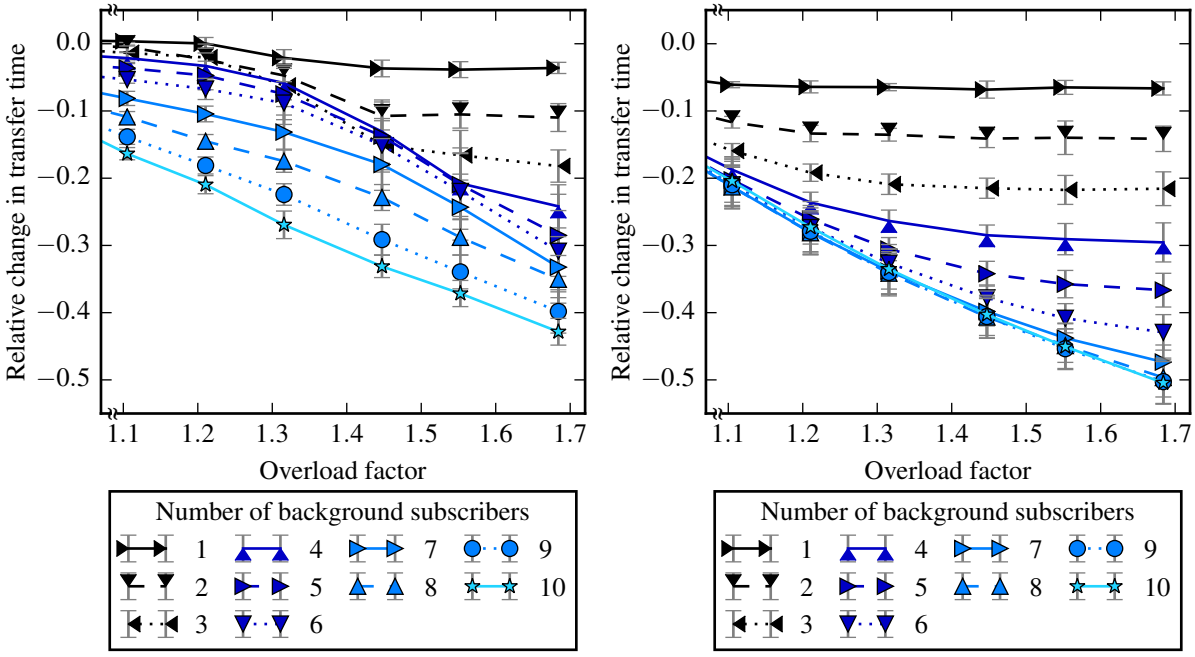
Figure 4.23: Foreground object transfer times for the BROAD topology, web traffic model and TCP Vegas-controlled background traffic



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.24: Foreground object transfer times for the BROAD topology, web traffic model and uTP-controlled background traffic



(a) For TCP Vegas-controlled background traffic

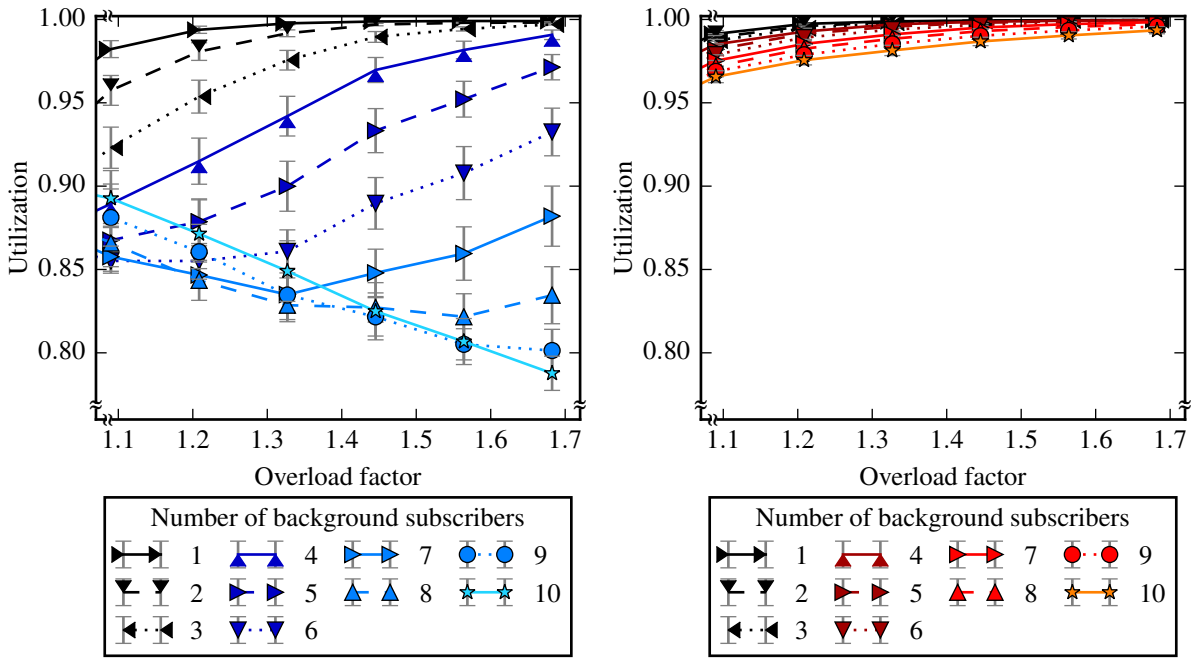
(b) For uTP-controlled background traffic

Figure 4.25: Relative change in foreground object transfer times introduced by applying RADICCO instead of the reference HFS for the BROAD topology and the web traffic model

Nevertheless, this is not the case as we show in the following. We additionally performed another evaluation of our results that evaluates the differences of the batch means. The approach for comparing expected responses of two systems is accepted in the community and the statistics theory provides a sound foundation [184]. In our case this approach allows to exploit the fact that both simulations have been carried out with the same sequence of pseudo random numbers, i.e. both simulations conduct the same sequence of object transfers (except that due to the delay-based traffic definition the more efficient approach, i.e. RADICCO, handles more transfers). We show the relative change of the batch means that is caused by applying RADICCO, i.e. any number smaller than zero means applying RADICCO causes the transfer times to decrease, i.e. a QoE improvement. Figure 4.25 shows the results of this evaluation approach, in Figure 4.25a for TCP Vegas-controlled background traffic, in Figure 4.25b for uTP-controlled traffic. These figures show that for foreground web transfers a benefit in terms of reduced transfer times is almost guaranteed if the reference HFS is replaced by a RADICCO scheduler. Only for a single background subscriber and low overload a negligible disadvantage may occur. Moreover, these figures clearly show the about linear dependency of gain on both the share of background traffic as well as the amount of overload.

4.6.2 Bottleneck Utilization

We also examine the bottleneck utilization for this scenario. While we find no significant difference between RADICCO and the reference HFS in case of TCP Vegas-controlled background traffic, we again find significant underutilization in case of uTP-controlled background traffic. We therefore show only the results for uTP-controlled background traffic in Figure 4.26, in



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.26: Utilization for the BROAD topology, web traffic model and uTP-controlled background traffic

Figure 4.26a if applying RADICCO and in Figure 4.26b if applying the reference HFS. Again, we see already known patterns. For low background traffic shares, the utilization increases with increasing offered load, but for high background shares (seven and eight background subscribers) the utilization first decreases and then rises again. For the highest evaluated background shares, i.e. nine and ten background subscribers, we mostly see a decrease in utilization with increasing load. The explanation is the same: On the one hand, with these highly bursty loads uTP connections frequently fall into their hibernation mode and therefore do not exploit available bandwidth during that time. This happens more often the higher the overload is and its effect on the overall utilization is the higher the more background subscribers exist. On the other hand, subscribers transmitting the updates traffic model using a foreground CCA, i.e. ten minus the background subscribers in this scenario, have a smoothing effect on the system since they have a non-empty buffer at least most of the time. Therefore, they use any bandwidth allocated to them and result in an increased utilization.

4.6.3 Scenario Conclusion

Also for web traffic applying RADICCO results in a significant benefit as clearly visible in Figure 4.25. Again, especially for uTP-controlled background traffic there is a price payed in terms of underutilization of the bottleneck, although not as large as for VoD traffic.

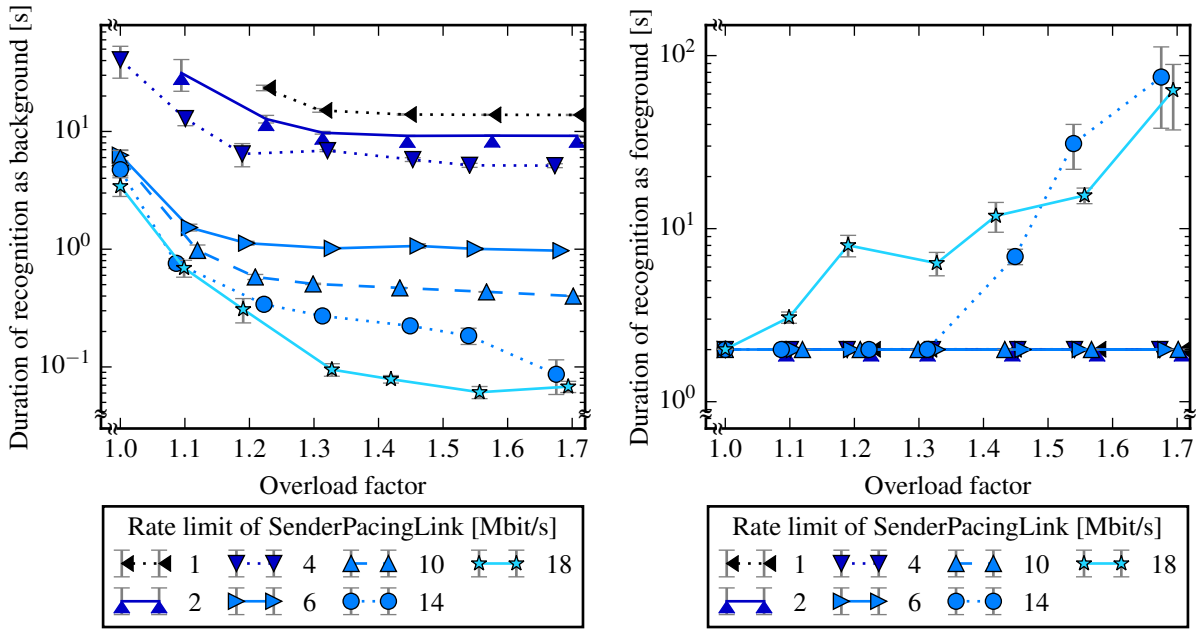
4.7 Performance for Otherwise Rate-limited Greedy Traffic

For the evaluation of otherwise rate-limited greedy traffic, we use the simulation model presented in Section 4.2.2 with the BROAD topology. We model nineteen subscribers to receive traffic according to the software updates traffic model, among these one to ten background subscribers. We configure the twentieth subscriber with a lower delivery speed by reducing the capacity of the respective SenderPacingLink (see Figure 4.3). Therefore, the traffic passing this link may be affected by queuing at two locations: At the buffer at the ingress of the SenderPacingLink and at the BNG scheduler. If the effective bandwidths at these two places differ significantly, e.g. in this scenario the rate limit is much lower than the assigned bandwidth at the scheduler, a queue virtually never builds up at the faster link, i.e. in the example at the BNG scheduler. In contrast, if both speeds are about the same on average (the scheduler's rate varies), queues may build up at both places at times, resulting in large packet delays.

RADICCO is expected to result in big queuing delays for this type of traffic (see Section 3.8.5.3). RADICCO is designed to not scale the buffer with the target rate, as argued in Section 3.8.5.2. Since the buffer equals 100 ms at the access link's speed of 20 Mbit/s, it equals 1.000 ms at 2 Mbit/s and 2.000 ms at 1 Mbit/s. More relevant than the delay equivalent to the total buffer is the delay equivalent to the configured foreground detection threshold of 35 % buffer usage. It corresponds to 700 ms for 1 Mbit/s, the minimum background target rate. However, the effective rate during overload mostly consisting of foreground traffic may be significantly lower than the target rate. If the configured overload factor directly applies to the formula given in Equation 3.37, the maximum queuing delay for the highest overload of 1.69 is to be expected to reach 1.183 s.

These simulations depict the worst-case in terms of the arriving traffic since the model results in the incoming traffic at the BNG not only arriving with rates persistently lower than the access link capacity, but at precisely constant rates. This is true for any duration during which the SenderPacingLink is the bottleneck. Since the SenderPacingLink is also equipped with 100 ms worth of drop-tail buffer, the foreground CCA keeps a queue in that buffer as long as this link is the bottleneck. So, after the slow start phase, this link's queue should only run empty if the scheduler serves this subscriber with a lower rate than the capacity of that link. At all other times, the SenderPacingLink acts as a traffic shaper, ensuring equidistant arrivals at the BNG. We are aware that this is a far stricter traffic model than e.g. applicable to today's video conferencing traffic. Today's video conferencing systems use Variable Bit Rate (VBR) encoders, and often send bursts of data. So, an actual video conference service is most likely recognized as foreground subscriber for a substantially higher time share than measured in these simulations. Nevertheless, the used configuration captures the worst case in that respect, so our results provide a lower boundary for potential performance.

For the examined traffic, several metrics are of interest: First, there is packet waiting time at the scheduler that impacts the packet one-way delay which is the crucial metric from service perspective if the greedy traffic is assumed representing a streaming session. We will look into the mean as well as the measured maximum waiting time although the latter has no statistical significance. Second, we examine the allocated bandwidth. For most cases, it should be equal to the limit rate. Only if the average allocated foreground subscriber rate at high overload is below the rate limit, the allocated rate may be lower than the limit rate. Before we present the



(a) Duration of background recognition

(b) Duration of foreground recognition

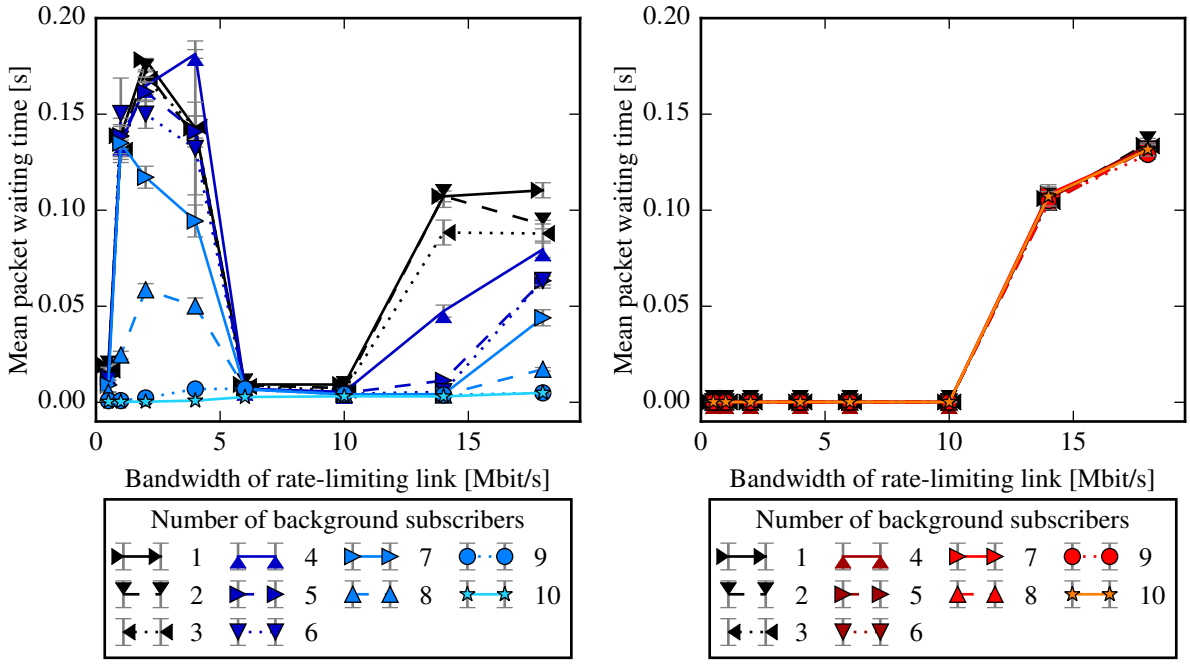
Figure 4.27: Duration of stable recognition of rate-limited traffic for the BROAD topology and one uTP-controlled background subscriber

results on traffic metrics, we evaluate our assumption of oscillating behavior (see Figure 3.6) by evaluating the durations during which RADICCO recognizes the rate-limited traffic as foreground or background traffic.

4.7.1 Phases of Constant Recognition

We evaluated the behavior of RADICCO regarding one rate-limited greedy flow for different loads at an overload factor of 1.69 and different rate limits. Since a high amount of background traffic may only diminish the (negative) effect we expect, we only configure one background subscriber and eighteen not limited foreground subscribers, all receiving traffic according to the updates traffic model.

We show the durations between recognition changes for the rate-limited traffic in Figure 4.27, in Figure 4.27a the mean durations of phases of background recognition, Figure 4.27b the mean durations of phases of foreground recognition. In not one simulation the rate-limited traffic is consistently recognized as foreground, there are even three parameterizations with consistent recognition as background, namely 1 Mbit/s rate-limited traffic at load levels 1.0 and 1.1 and 2 Mbit/s rate-limited traffic at load level 1.0. Moreover, simulations with 0.5 Mbit/s rate-limited traffic are also recognized as background traffic as expected. Several insights are gained from these figures: First, most parameterizations result in foreground durations of almost exactly two seconds, the configured background detection threshold. Only the rather high rate limits 18 Mbit/s and 14 Mbit/s manage to fill the buffer sufficiently for a longer time than this threshold. For the 14 Mbit/s rate-limited traffic this only applies for rather high levels of overload. Second,



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.28: Packet waiting times in the scheduler's queue for rate-limited traffic in the BROAD topology, uTP-controlled background traffic and a overload factor of 1.69

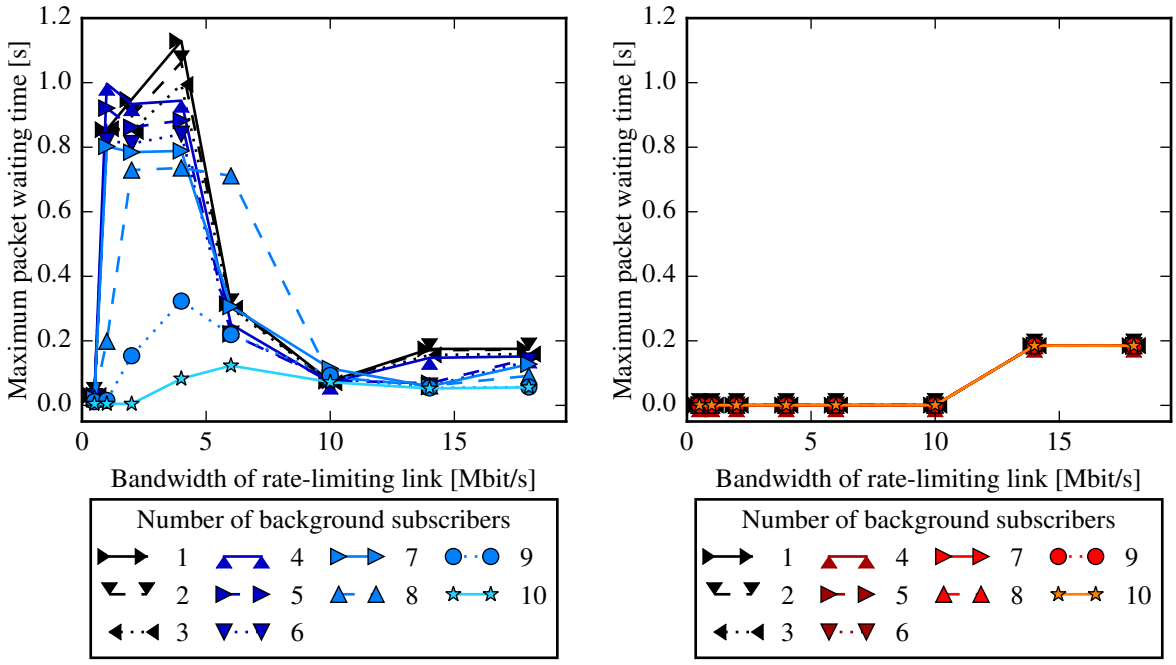
all measurements show small to close to zero confidence intervals, indicating a very strict and static oscillation pattern. This applies to the small and medium rate limits for medium and high loads, i.e. greater than 1.3. Third, with increasing load the background durations decrease for all rate limits, and this decrease gets less steep the higher the load. Note also the different scales of the two figures and consider the range of resulting ratios of foreground- to background time of the different parameterizations: While the 18 Mbit/s rate-limited traffic is recognized as background for only about $^1/_{1000}$ of the time at a load of 1.69, the ratio for 4 Mbit/s rate-limited traffic at load 1.0 is about $^{20}/_1$.
So our expectation of an oscillating behavior is met, yet on different levels for the different parameterizations. We will analyze its consequences in the following.

4.7.2 Waiting Time

In this section, we present results of two types of parameterizations: First, we evaluate if the resulting packet delay depends from the number of background subscribers and the offered load. Second, we evaluate the impact of the limit rate for different offered loads on the packet delay.

4.7.2.1 Waiting Time for Different Rate Limits

First, we evaluate a scenario with overload factor 1.69, the highest value we consider in all our simulations, and vary the number of background subscribers. Figure 4.28 shows the mean packet waiting time in the BNG scheduler's queue for the rate-limited flow, either when applying



(a) Applying RADICCO

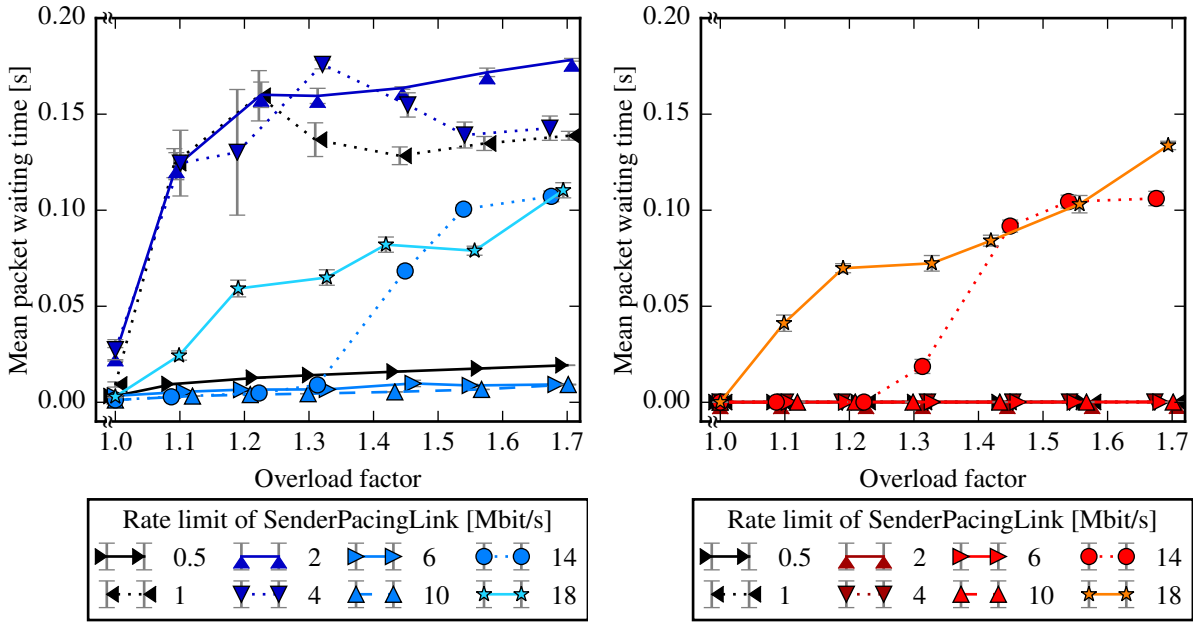
(b) Applying the reference HFS

Figure 4.29: Maximum packet waiting times in the scheduler's queue for rate-limited traffic in the BROAD topology, uTP-controlled background traffic and a overload factor of 1.69

RADICCO, Figure 4.28a, or when applying the reference HFS, Figure 4.28b. Here, we show only the results for uTP-controlled background traffic since the results for TCP Vegas-controlled are less interesting: The waiting times for all rate limits are very like the waiting times for the case of only one uTP-controlled background subscriber.

The mean packet waiting time when applying RADICCO is negligible for a rate limit of 0.5 Mbit/s, mostly high for small rate limits between 1 Mbit/s and 4 Mbit/s and for high rate limits, but close to zero for medium values. In contrast, the reference HFS—as expected—results in almost no waiting times for rate limits up to 10 Mbit/s and increasing waiting times for the high rate limits (14 and 18 Mbit/s). This behavior of the unmodified HFS meets our expectations since it always allocates the fair share, which at the examined loads are greater than 10 Mbit/s. Therefore, for rate limits greater or equal 10 Mbit/s the effective bottleneck—and the only queue—is at the rate-limiting link. Applying RADICCO changes this: If a subscriber is recognized as background subscriber, its target rate will be reduced to the minimum, in our parameterization 1 Mbit/s. The effective rate is even smaller in this scenario since the configured overload factor of 1.69 must be expected to result in RADICCO operating in Heavy Overload state most of the time. For 0.5 Mbit/s rate limited traffic, the scheduler does not constitute the bottleneck despite the further reduced effective rate, so the queuing delay is negligible. Yet, for the higher rate limits, this does not hold. Here we already can see the negative effect of the combination of the untouched buffer limit designed for the access link nominal capacity and the reduced effective rate.

This effect becomes even more impressive when looking into maximum waiting times. These are only single values and have no statistical significance but give a clear impression of which ranges



(a) Applying RADICCO

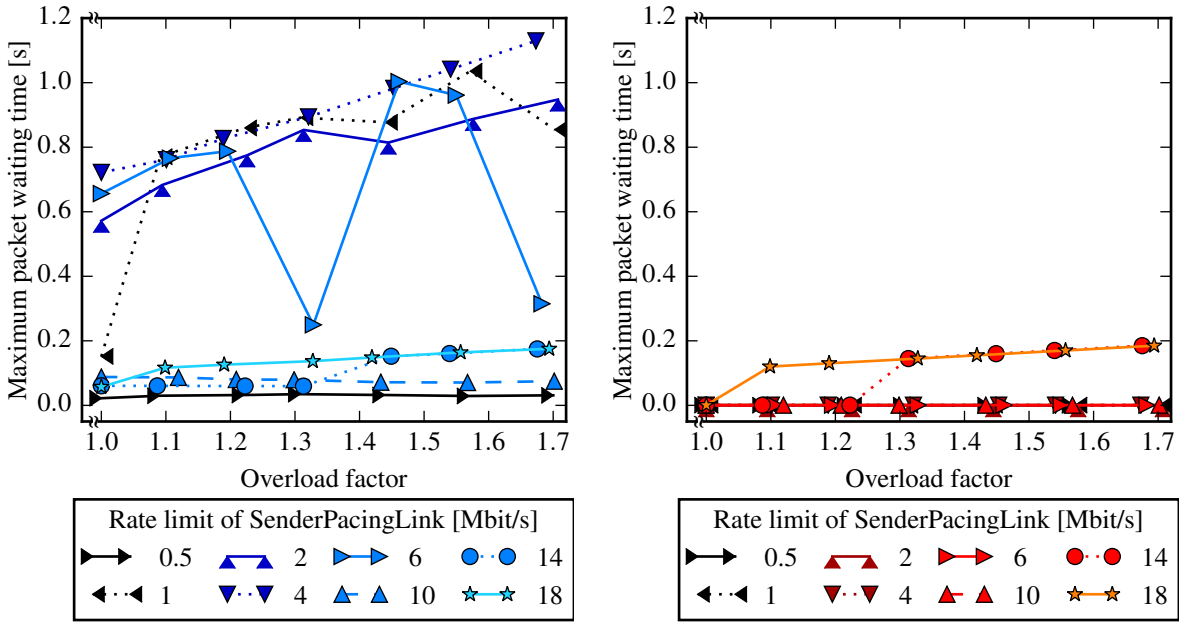
(b) Applying the reference HFS

Figure 4.30: Packet waiting times in the scheduler's queue for rate-limited traffic in the BROAD topology and one uTP-controlled background subscriber

waiting times may reach. Figure 4.29 shows the maximum packet waiting times, Figure 4.29a when applying RADICCO, Figure 4.29b when applying the reference HFS. Again, we omit the results for TCP Vegas-controlled background traffic since it is very like the results for one uTP subscriber. The huge waiting times for low rate limits are the direct consequence of reducing the target rate down to 1 Mbit/s while leaving the buffer size and the foreground detection threshold untouched. The reason for the effect of more uTP background subscribers reducing the waiting time lies in uTP's hibernation behavior that in that cases reduces the actual load.

So, if the bottleneck is at the scheduler and not at the rate-limiting link, the reduced serving rate implies a common maximum waiting time for all rate limits that result in such a state. Therefore, for most numbers of background subscribers, there is plateau for the rate limits 1–4 Mbit/s. These plateaus differ depending on the number of background subscribers since the more background traffic is recognized, the higher the final rate. As explained in Section 3.7.7, the final rate is obtained by sharing the capacity among all target rates. Assuming the subscriber in focus at the time has a target rate equal to the minimum 1 Mbit/s of a background subscriber, the final rate may be about divided by the current overload factor of 1.69 if all other subscribers are foreground subscribers. If nine, i.e. about half of the other subscribers is also at that minimum rate, the effective rate at a load of 1.69 is even higher than 1 Mbit/s.

Nevertheless, the results presented only concern a very high overload factor of 1.69. We therefore also looked into the relation between overload and induced waiting time.



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.31: Maximum packet waiting times in the scheduler's queue for rate-limited traffic in the BROAD topology and one uTP-controlled background subscriber

4.7.2.2 Waiting Time for Different Loads

In the simulation presented in the following we always used just one background subscriber, as it is the worst case as seen and explained above.

The mean packet waiting time depending on the offered load in shown in Figure 4.30, again for RADICCO, Figure 4.30a and the reference HFS, Figure 4.30b. For the high rate limits, i.e. 14 Mbit/s and 18 Mbit/s, not only the behavior for high load, as seen before, but also for all loads is not substantially different from the waiting time induced by the reference scheduler. For rate limits 6 and 10 Mbit/s there seems to be no significant difference in waiting times between RADICCO and the reference HFS on the first glance. Nevertheless, there are differences: While the reference HFS does not induce significant waiting time at all (< 0.2 ms for all loads), applying RADICCO increases this waiting time to a few milliseconds, at the highest measured load of about 9 ms. Such an increase is not significant in today's Internet since a more generously dimensioned buffer at the bottleneck easily induces a greater impact. The mean waiting time for 6-Mbit/s-limited traffic is a bit higher than for the traffic with higher rate limits and its confidence intervals are non-negligible except for overload factors 1.32 and 1.69. The results for the lowest rate limit of 0.5 Mbit/s show a rather low waiting time that increases with increasing load from 3 ms to a maximum of about 20 ms. The most interesting and alarming measurements are the ones for the even lower rate limits of 1–4 Mbit/s. We see a very high and slightly increasing large average waiting time for any overload.

The corresponding maximum waiting times are also high for load levels above 1.0, as can be seen in Figure 4.31. We show the measurements for RADICCO in Figure 4.31a and for the reference HFS in Figure 4.31b. Again, we can identify four groups of patterns:

- Traffic with high rate limits that show minor impact that increases linearly with the overload.
- Traffic with medium rate limits (10 Mbit/s only) that are not substantially impacted.
- Traffic with low rate limits (1–6 Mbit/s) that are heavily impacted.
- Traffic with the lowest rate limit (0.5 Mbit/s) that is not substantially impacted.

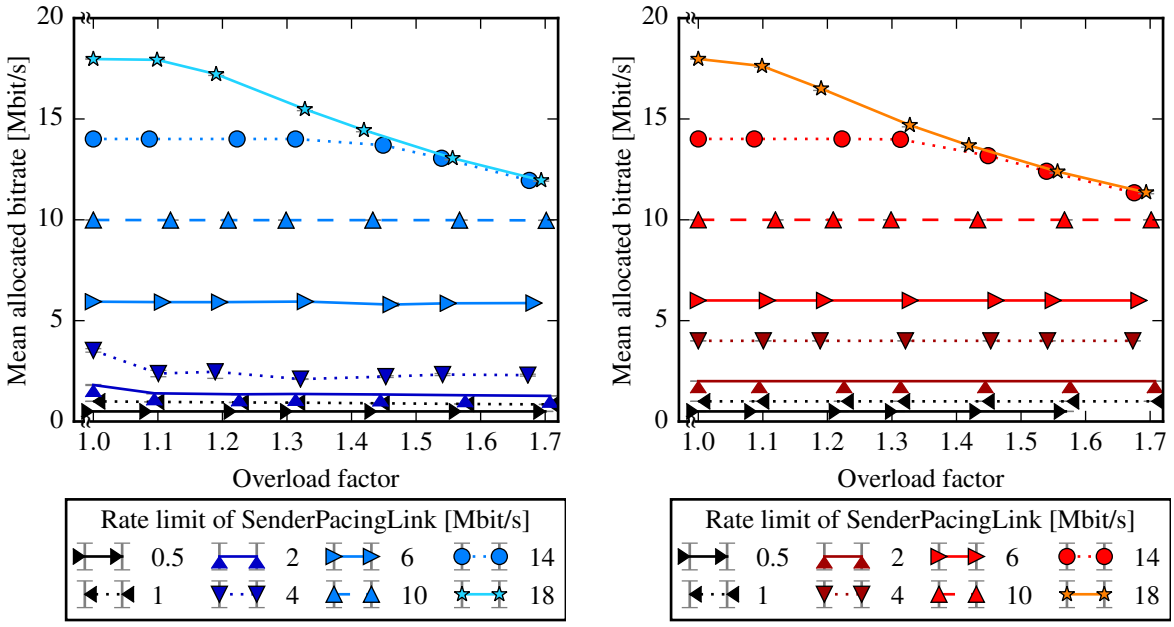
Interestingly the 6-Mbit/s-limited traffic does not seem to belong to the group of the 10-Mbit/s-limited traffic but to the group of lower rate limits, in contrast to the results on mean packet waiting times.

We start detailed discussion with the highest rate limits of 18 and 14 Mbit/s. They both show similar shapes for RADICCO as for the reference HFS since for loads higher than some overload threshold the hierarchical scheduler defines the bottleneck and not the rate-limited link. Nevertheless, for the 14 Mbit/s rate-limited traffic, this threshold seems to be shifted to a higher load for RADICCO compared to the reference HFS. Unfortunately, this probably is just a consequence of a random peak in load for the HFS simulation: for the configured offered load of 1.32 on average every subscriber should receive 15.15 Mbit/s. Moreover, the elastic traffic model with idle times results in higher instant shares most of the time.

The second group consists of the 10-Mbit/s-rate-limited traffic only. For this traffic, the bottleneck is never located at the BNG interface. Due to the rate variations induced by RADICCO, queuing occurs in the scheduler and therefore the queuing delay is significantly higher than for the reference HFS. The maximum values are at about 80 ms for any simulated load.

The third group is defined by traffic, for which the bottleneck alternates with substantial durations between both places. Here, the oscillation described in Section 3.8.5.3 applies and we see the expected huge waiting times. Interestingly, for the 6-Mbit/s-rate-limited traffic the RADICCO-enhanced BNG scheduler is not the bottleneck most of the time, so no significant queue builds up as seen in the mean delay shown in Figure 4.30a. Nevertheless, at rare constellations, i.e. many active foreground subscribers, the bottleneck shifts to the BNG, thus causing similar problems as for the lower rate limits. In Figure 4.31a we see that we captured such events for most overload factors but not for overload factors 1.32 and 1.69.

The fourth group consists of traffic whose rate limit is so low that the BNG never is the bottleneck. Nevertheless, applying RADICCO implies less constant scheduling so the waiting time increases significantly from below 1 ms to a maximum of 35 ms. With this value, the overall OWD does not exceed 85 ms, so it remains well below the 150 ms maximum OWD recommended by the ITU [G.114].



(a) Applying RADICCO

(b) Applying the reference HFS

Figure 4.32: Throughput of rate-limited traffic for the BROAD topology and one uTP-controlled background subscriber

4.7.3 Bandwidth Allocation

In Figure 4.32 the mean allocated bandwidth is plotted, in Figure 4.32a for RADICCO, in Figure 4.32b for the reference HFS. Clearly, the reference HFS provides an actual perfect allocation: Since all subscribers have same weight corresponding to their equal access link capacity, every subscriber receives the maximum of incoming rate (here limited by the SenderPacingLink) and the current fair share. Since the traffic source cannot catch up after a time of a lower effective rate at the BNG scheduler, average throughput decreases earlier than the average offered load might suggest. For instance, the 18 Mbit/s-rate-limited subscriber does not reach 18 Mbit/s at load 1.0 but just 17.97 Mbit/s and at load 1.1 just 17.93 Mbit/s. Note also the barely visible confidence intervals for the reference HFS.

Comparing the achieved throughputs of RADICCO with the reference HFS, we find no significant difference for the higher three rate limits and even for the 6 Mbit/s rate-limited traffic the difference is small (less than 3 %). In contrast, lower rate limits result in significantly reduced allocations. For instance, the traffic limited to 4 Mbit/s receives between 2.1 and 2.5 Mbit/s for overload factors of 1.1 and higher and traffic limited to 2 Mbit/s receives between 1.27 and 1.4 Mbit/s for overload factors of 1.1 and higher. The root cause for these low rates are the long durations recognized as background as shown already in Figure 4.27a. A rate of 6 Mbit/s suffices to leave background recognition after about one second, most of which the target rate is still well above 6 Mbit/s. Therefore, the average rate of such 6 Mbit/s rate-limited traffic is determined by the arrival rate and the minimum background target rate has not a big influence. For lower rates the situation is contrary: There are long durations of background recognition, therefore implying a low or even minimum target rate for the biggest part of these durations, resulting in

a much lower average target rate than the rate limit. These low target rates become effective at overload only and the higher the overload and the less background traffic is present in the aggregate, the more effective the target rates are. Since there is only one background subscriber, even low overload (of 1.1) or temporary overload (at stochastic load 1.0) results in an effective rate substantially lower than the respective rate limits. For the 1 Mbit/s rate-limited traffic the situation is basically the same but the minimum target rate is not lower than the rate limit, so the impact is much less severe. The lowest rate-limit of 0.5 Mbit/s in contrast receives almost perfectly its full rate, the largest deviation is 53 bit/s.

So, also the bandwidth allocation mirrors the effects of false background recognition: Rate limits that are not subject to false recognition or receive a rate lower than the effective minimum rate receive about the same rate as with the HFS. In contrast, the medium rates receive reduced throughput rate only.

4.7.4 Scenario Conclusion

The evaluation of rate-limiting traffic confirms our expectations and concerns. Even for low overload, applying RADICCO results in large queuing delays at the scheduler for traffic that arrives at rates much lower than the respective access link's bitrate. With the configured low minimum background target rate, for traffic arriving with about half the respective access link's bitrate or higher, there is no significant negative effect. Even for rate limits above 6 Mbit/s we found limited impact in form of sporadic large delays for the evaluated 20 Mbit/s access links.

These results cover almost the worst case for several reasons: First, we simulated partly high overload factors. Usual ISPs do not operate their networks with an overload greater than 1.2, as the measurements during peak load suggest (see Section 2.1.3). We also simulate higher overload factors since we aim to evaluate the capabilities and limits of RADICCO. Second, we simulated a constant bitrate bottleneck. Real sender-limited traffic mostly has a variable bitrate, e.g. video conferencing calls. Bottlenecks at other links in the network are usually shared with other flows, also resulting in varying throughput. Both scenarios result in bursts of traffic arriving at the BNG, which would trigger the traffic being recognized as foreground traffic. Third, we show that traffic below the minimal effective rate ($\frac{bg^{min}}{OLF}$) is protected from excessive queuing delays. We set this value extremely low for the same motivation we used high overload factors. For instance, configuring bg^{min} to 10 Mbit/s would protect all normal VoIP and real-time video conferences for usual overload factors.

Nevertheless, for a product based on RADICCO, the issue of handling otherwise rate-limited traffic can and should be solved by introducing a detection mechanism for this rare third traffic type as outlined in Section 3.8.5.3.

4.8 Evaluation Summary

The evaluation of RADICCO shows that RADICCO fulfills all qualitative objectives in praxis. We summarize the result of the evaluation in Table 4.1.

Table 4.1: Evaluation summary

Objective	Abstract rating	Comment
Network neutrality	√	Mechanism is controlled by end hosts and treats all services the same.
Sufficient efficiency	√	In praxis $O(1)$, although theoretical complexity. $O(N)$
Smooth Rate Allocations for Foreground Traffic	√	At least as smooth as HFS except when allocating bandwidth from background to new foreground.
QoE improvement	++	QoE improvement for foreground traffic with a bottleneck at the BNG. Extension required for otherwise rate-limited traffic.
High bottleneck utilization	+	Depends on background CCA. For uTP low for bursty traffic.
Fairness among foreground subscribers	+++	Close to perfect.
Fairness among background subscribers	-	Depends on background CCA. Interaction of control loops of CCAs and RADICCO may result in unfairness.

The simulative performance evaluation shows that, if the BNG is the bottleneck, RADICCO improves QoE for all examined traffic models. In particular, we show that RADICCO mechanisms work on both, the leaf node level but if the resource reallocation is required at a higher hierarchy level. The benefit can be distinguished in two levels: First, RADICCO results in overload not deteriorating the foreground traffic's QoE up to a certain level of overload. There often is an approximately linear relation between the share of background traffic and the limiting level. Second, beyond this level of overload the QoS decreases. Transfer times increase about linearly with the elastic load, but the QoS is substantially improved compared to the HFS. The slope of the increase depends on the share of background traffic, more background traffic results in a lower slope. At all loads and with any share of background traffic, RADICCO maintains a high fairness among foreground subscribers. Nevertheless, performance differs depending on the CCA used by the background traffic: If uTP is used, benefit and fairness, among both foreground and background subscribers, are greater than for TCP Vegas. The cause for this difference is the background detection mechanism reliably recognizing uTP but often recognizing TCP Vegas-controlled flows as foreground. For uTP-controlled background traffic RADICCO often results in significantly lower bottleneck utilization compared to the HFS. This is not the case for TCP Vegas. The reason for this difference is the behavior of uTP that often reacts with long hibernation phases to the rate changes introduced by RADICCO.

If the incoming traffic is rate-limited at another location, RADICCO may induce large packet delay which is prohibitive for several services that base on traffic that is rate-limited by the sender, e.g. (real-time) VoIP and video conferencing. The induced packet delay scales with the relation of the subscriber's capacity $C_{i,j}$ to his minimum target rate $bg_{i,j}^{min}$. Therefore, for high minimum target rates the effect is less severe, so the minimum target delay can be used to

adjust the maximum packet delay assuming the overload is known. Furthermore, a deploying ISP can differentiate the extent of this effect per subscriber. Nevertheless, we assume that any product implementing RADICCO should implement an extension mitigating this behavior, see Section 3.8.5.3 for possible approaches.

5 Conclusion and Outlook

This section provides a short summary of this thesis, draws conclusions and provides an outlook on future work.

In this thesis, we introduced the novel concept of Rate Adaptation Considering Traffic Differentiation by Congestion Control during Overload (RADICCO), which uses knowledge on transport layer mechanisms to improve a hierarchical scheduler's MAC layer function. RADICCO is intended for deployment at the BNG downstream hierarchical scheduler, which is the bottleneck for most downstream traffic due to the typical design of ISP network topologies.

Chapter 2 gave an overview on the relevant background. Since RADICCO combines packet scheduling with CC behavior and is designed for the deployment at BNGs, this chapter covered access networks and their role, packet schedulers, congestion and transport layer CC.

In Chapter 3, we gave a motivation for this thesis, a description of the problem to be approached in it and presented the concept of RADICCO. In continuation, we presented our objectives and related work, provided a detailed algorithm description and discussed design decisions and alternatives.

In Chapter 4, we evaluated RADICCO according to the defined qualitative design goals and quantitative performance objectives. We found that RADICCO meets the objectives of network neutrality and smooth rate allocations for foreground traffic. The practical computational complexity is acceptable and allows implementation. Therefore, we considered the qualitative objectives met by RADICCO. The performance evaluation was carried out by simulations incorporating unmodified wide-spread real-world CCA implementations of TCP Cubic for foreground traffic and TCP Vegas and uTP for background traffic. The simulative performance evaluation showed that RADICCO improves QoE for all examined traffic models if the BNG is the bottleneck. In particular, we showed that RADICCO's mechanisms achieve a beneficial bandwidth reallocation on both levels, on the AN level and on the AGS level. Regarding QoE, the performance evaluation shows two correlations:

1. RADICCO avoids deterioration of QoE despite overload up to a certain limit. The more background traffic exists in the traffic mix, the higher the limit of overload, which RADICCO can accept without QoS deterioration.
2. At load beyond that limit, the QoE decreases, but substantially less than for a HFS. The slope also depends on the share of background traffic, more background traffic results in a lower deterioration.

The bottleneck utilization achieved by RADICCO depends heavily on the CCA of the background traffic. Defensive CCAs such as BitTorrent's uTP partly react to the rate changes implemented by RADICCO with long phases of inactivity, causing low utilization despite increased foreground rates. The fairness among foreground subscribers achieved by RADICCO is high at all loads and with any share of background traffic. Yet, RADICCO has a drawback for incoming traffic that has no bottleneck at the BNG because it is rate-limited at another location. For this rare type of traffic, RADICCO may induce large packet delay depending on scenario and parameterization.

In the following, we analyze potential impact for concerned parties and their possible reactions. RADICCO particularly concerns:

- The deploying ISP.
- The users in the subscriber households.
- OS designers, application developers and service providers.

A deploying ISP substantially benefits from RADICCO. RADICCO allows to postpone costly investments into infrastructure upgrades and to accept daily overload since RADICCO shifts the limits of offered load that can be handled without QoE deterioration. Moreover, RADICCO reduces the QoE deterioration beyond these limits, so the pressure for investments is reduced for such overload. If the ISP delays upgrades, RADICCO results in higher average utilization due to the avoided bandwidth reductions of adaptive services such as DASH-like VoD. The average utilization also becomes smoother, easing routing and traffic engineering in the ISP's metro and core networks. Moreover, these benefits can be achieved with unchanged or not significantly changed BNG devices. This is a strong contrast to DPI-based overload management that requires substantial computational resources and therefore expensive devices.

Users, and thus the subscribers, also benefit from RADICCO being deployed. For most situations, foreground traffic is transferred faster or at least as fast as with HFS, and QoE is generally improved. Nevertheless, we do not expect a change in users' behavior, since most transfers are not triggered and controlled by a human user directly but by an application or the OS of a device.

For OS designers, application developers and service providers, a deployment of RADICCO increases the achievable QoE, yet for these players the reason for the increased network performance does not matter. For service providers, peak loads may change: For foreground services, the load during the network's peak periods increases due to RADICCO allocating more bandwidth to their receivers. For background services, the load during these periods decreases but after them increases correspondingly. Apart from that, the incentive situation changes. Since OS designers define one solution that is applied to many devices, the design is based on a statistical view on all deployed systems. From this perspective, RADICCO introduces the opportunity for subscribers to mutually benefit from using traffic differentiation by CC on average. Thus, RADICCO adds incentive for OS designers to use traffic differentiation by CC, although we consider the competition for Internet bandwidth of the networked devices within a subscriber's household providing sufficient incentive. For application developers and service providers that offer both foreground services as well as background services, the same argument applies. Nevertheless, its importance decreases with decreasing prevalence of the respective application or service.

After this thesis introduced and evaluated the novel concept of RADICCO, there are still numerous questions that merit further investigation. We recognize three main purposes:

- Enhance RADICCO to detect and handle otherwise rate-limited traffic.
- Evaluate and tune RADICCO by a prototype with real Internet traffic.
- Evaluate if RADICCO's concept can also be applied to other network bottlenecks, e.g. in cellular networks.

Enhancing RADICCO to detect and appropriately handle otherwise rate-limited traffic is required if RADICCO shall be configured with low minimum background target rates. This is attractive since low minimum background target rates achieve greater benefits for low shares of background traffic.

Real communication networks differ from our simulation by several properties, that may impact the behavior of RADICCO as well as the CCAs and may imply a need for re-calibrating RADICCO parameters. Therefore, measuring a prototype with real Internet traffic would be desirable. Potential causes for deviating behaviors are the variation of available BE capacity caused by priority traffic and the jitter today's network cards produce due to offloading and interrupt coalescing.

Finally, it is interesting how RADICCO performs when applied to other network bottlenecks, in particular the packet gateway of a cellular network. Up to now, this is no use case since most high volume and background transfers of mobile devices are carried out using local WiFi connections, which access the Internet via wired access links.

Bibliography

[1] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. QoS's Downfall: At the bottom, or not at all! In *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?*, pages 109–114, New York, NY, USA, 2003. ACM. URI: http://doi.acm.org/10.1145/944592.944594, doi:http://doi.acm.org/10.1145/944592.944594.

[2] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *Internet Measurement Conference*, pages 43–56, 2007.

[3] Broadband Internet Technical Advisory Group, Inc. Real-time Network Management of Internet Congestion. Technical report, Broadband Internet Technical Advisory Group, Inc., Oct 2013. URI: https://www.bitag.org/documents/BITAG_-_Congestion_Management_Report.pdf.

[4] Nokia Networks. 7750 Service Router. Web site. Retrieved 2017-01-18. URI: https://networks.nokia.com/products/7750-service-router.

[5] Cisco Systems, Inc. Cisco ASR 9000 Series Aggregation Services Routers. Retrieved 2017-01-18. URI: http://www.cisco.com/c/en/us/products/routers/asr-9000-series-aggregation-services-routers/index.html.

[6] Akamai Technologies, Inc. Akamai's State of the Internet Q2 2016 Report. Technical report, Akamai Technologies, Inc., September 2016.

[7] Dialog Consult / VATM. 18. TK-Marktanalyse Deutschland 2016. Technical report, VATM, October 2016. Retrieved 2016-11-10 from http://www.vatm.de/vatm-marktstudien.html.

[8] W. Coomans, R. B. Moraes, K. Hooghe, A. Duque, J. Galaro, M. Timmers, A. J. van Wijngaarden, M. Guenach, and J. Maes. XG-FAST: Towards 10 Gb/s copper access. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 630–635, Dec 2014. doi:10.1109/GLOCOMW.2014.7063503.

[9] K. A. Noll. Hybrid Fibre-Coaxial Networks: Technology and Challenges in Deploying Multi-Gigabit Access Services, Presentation on the 7th NANOG on the road meeting, June 2015. Retrieved 2016-12-21.

[10] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 90–102, New York, NY, USA, 2009. ACM. URI: http://doi.acm.org/10.1145/1644893.1644904, doi:10.1145/1644893.1644904.

[11] K. Fukuda, K. Cho, and H. Esaki. The Impact of Residential Broadband Traffic on Japanese ISP Backbones. *SIGCOMM Comput. Commun. Rev.*, 35(1):15–22, 2005. URI: http://doi.acm.org/10.1145/1052812.1052820, doi:10.1145/1052812.1052820.

[12] L. Quan, J. Heidemann, and Y. Pradkin. When the Internet Sleeps: Correlating Diurnal Networks with External Factors. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 87–100, New York, NY, USA, 2014. ACM. URI: http://doi.acm.org/10.1145/2663716.2663721, doi:10.1145/2663716.2663721.

[13] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, P. D. Torino, and D. Rossi. Experiences of Internet traffic monitoring with tstat. *IEEE Network*, 25(3):8–14, May 2011. doi:10.1109/MNET.2011.5772055.

[14] K. Lukas, A. Marx, B. O. Schöttler, and C. Sudhues. „dienstequalität von breitbandzugängen ii“ - studie im auftrag der bundesnetzagentur. Technical report, zafaco GmbH, jun 2014. URI: http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Breitband/Breitbandmessung/Qualitaetsstudie/AbschlussberichtQualitaetsstudie2013.pdf.

[15] FCC's Office of Engineering and Technology and Consumer and Governmental Affairs Bureau. 2015 measuring broadband america - fixed broadband report. Technical report, FCC's Office of Engineering and Technology and Consumer and Governmental Affairs Bureau, 2015.

[16] T.-Y. Tsai, Y.-L. Chung, and Z. Tsai. Introduction to packet scheduling algorithms for communication networks. *Communications and Networking*, pages p263–271, 2010.

[17] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Symposium Proceedings on Communications Architectures and Protocols*, pages 1–12, New York, NY, USA, 1989. ACM. URI: http://doi.acm.org/10.1145/75246.75248, doi:10.1145/75246.75248.

[18] K. Nichols and V. Jacobson. Controlling Queue Delay. *Queue*, 10(5):20:20–20:34, 2012. URI: http://doi.acm.org/10.1145/2208917.2209336, doi:10.1145/2208917.2209336.

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993. doi:10.1109/90.251892.

[20] P. Lothberg. Terastream – a simplified ip network service delivery model. https://ripe67.ripe.net/archives/video/3/, October 2013. Retrieved 2016-12-21. URI: https://ripe67.ripe.net/archives/video/3/.

[21] P. L. Dordal. *An Introduction to Computer Networks, Release 1.8.22*. Department of Computer Science, Loyola University Chicago, 1.8.22 edition, July 2016.

[22] L. R. Tymes. Routing and Flow Control in TYMNET. *IEEE Transactions on Communications*, 29(4):392–398, Apr 1981. doi:10.1109/TCOM.1981.1095020.

[23] A. Fraser. Towards a Universal Data Transport System. *IEEE Journal on Selected Areas in Communications*, 1(5):803–816, Nov 1983. doi:10.1109/JSAC.1983.1145998.

[24] J. Nagle. On Packet Switches with Infinite Storage. *IEEE Transactions on Communications*, 35(4):435–438, Apr 1987. doi:10.1109/TCOM.1987.1096782.

[25] L. Kleinrock. Time-shared Systems: A Theoretical Treatment. *J. ACM*, 14(2):242–261, 1967. URI: http://doi.acm.org/10.1145/321386.321388, doi:10.1145/321386.321388.

[26] L. Kleinrock. *Queueing Systems*, volume II: Computer Applications. Wiley Interscience, 1976. (Published in Russian, 1979. Published in Japanese, 1979.).

[27] L. Kleinrock and R. R. Muntz. Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time Shared System. *J. ACM*, 19(3):464–482, 1972. URI: http://doi.acm.org/10.1145/321707.321717, doi:10.1145/321707.321717.

[28] L. Kleinrock, R. R. Muntz, and E. Rodemich. The processor-sharing queueing model for time-shared systems with bulk arrivals. *Networks*, 1(1):1–13, 1971. URI: http://dx.doi.org/10.1002/net.3230010103, doi:10.1002/net.3230010103.

[29] E. Gafni and D. Bertsekas. Dynamic control of session input rates in communication networks. *IEEE Transactions on Automatic Control*, 29(11):1009–1016, Nov 1984. doi:10.1109/TAC.1984.1103431.

[30] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *INFOCOM 99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1395–1403 vol.3, 1999. doi:10.1109/INFCOM.1999.752159.

[31] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Transactions on Networking*, 10(3):320–328, Jun 2002. doi:10.1109/TNET.2002.1012364.

[32] A. G. Fraser and S. P. Morgan. Queueing and framing disciplines for a mixture of data traffic types. *AT&T Bell Laboratories Technical Journal*, 63(6):1061–1087, July 1984. doi:10.1002/j.1538-7305.1984.tb00114.x.

[33] C. Y. Lo. Performance analysis and application of a two-priority packet queue. *AT&T Technical Journal*, 66(3):82–99, May 1987. doi:10.1002/j.1538-7305.1987.tb00213.x.

[34] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. *SIGCOMM Comput. Commun. Rev.*, 20(4):19–29, 1990. URI: http://doi.acm.org/10.1145/99517.99525, doi:10.1145/99517.99525.

[35] L. Zhang. *A new architecture for packet switching network protocols*. PhD thesis, 1989. URI: http://hdl.handle.net/1721.1/14184.

[36] A. K. J. Parekh. *A Generalized Processor Sharing Approach to Flow Control In Integrated Services Networks*. PhD thesis, 1992. URI: http://www.tecknowbasic.com/thesis.pdf.

[37] J. Bennett and H. Zhang. WF^2Q: worst-case fair weighted fair queueing. In *INFOCOM 96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 1, pages 120–128 vol.1, Mar 1996. doi:10.1109/INFCOM.1996.497885.

[38] J. C. R. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. In *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 143–156, New York, NY, USA, 1996. ACM. URI: http://doi.acm.org/10.1145/248156.248170, doi:10.1145/248156.248170.

[39] D. Stephens, J. Bennett, and H. Zhang. Implementing scheduling algorithms in high-speed networks. *Selected Areas in Communications, IEEE Journal on*, 17(6):1145–1158, Jun 1999. doi:10.1109/49.772449.

[40] S. Suri, G. Varghese, and G. Chandranmenon. Leap forward virtual clock: a new fair queuing scheme with guaranteed delays and throughput fairness. In *INFOCOM 97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 2, pages 557–565 vol.2, Apr 1997. doi:10.1109/INFCOM.1997.644506.

[41] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, Jun 1996. doi:10.1109/90.502236.

[42] G. Chuanxiong. SRR: An O(1) Time Complexity Packet Scheduler for Flows in Multi-service Packet Networks. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 211–222, New York, NY, USA, 2001. ACM. URI: http://doi.acm.org/10.1145/383059.383076, doi:10.1145/383059.383076.

[43] L. Lenzini, E. Mingozzi, and G. Stea. Aliquem: a novel DRR implementation to achieve better latency and fairness at O(1) complexity. In *Quality of Service, 2002. Tenth IEEE International Workshop on*, pages 77–86, 2002. doi:10.1109/IWQoS.2002.1006576.

[44] S. Ramabhadran and J. Pasquale. Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 239–250, New York, NY, USA, 2003. ACM. URI: http://doi.acm.org/10.1145/863955.863983, doi:10.1145/863955.863983.

[45] X. Yuan and Z. Duan. FRR: a proportional and worst-case fair round robin scheduler. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 831–842 vol. 2, March 2005. doi:10.1109/INFCOM.2005.1498314.

[46] C. Sun, L. Shi, C. Hu, and B. Liu. DRR-SFF: A Practical Scheduling Algorithm to Improve the Performance of Short Flows. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 13–13, June 2007. doi:10.1109/ICNS.2007.54.

[47] S. Bakiras, F. Wang, D. Papadias, and M. Hamdi. Vertical dimensioning: A novel implementation for efficient fair queueing . *Computer Communications*, 31(14):3476 – 3484, 2008. URI: http://www.sciencedirect.com/science/article/pii/S0140366408003599, doi:http://dx.doi.org/10.1016/j.comcom.2008.06.008.

[48] J. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *Networking, IEEE/ACM Transactions on*, 5(5):675–689, Oct 1997. doi:10.1109/90.649568.

[49] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM 94. Networking for Global Communications., 13th Proceedings IEEE*, pages 636–646 vol.2, Jun 1994. doi:10.1109/INFCOM.1994.337677.

[50] M. Karsten. Approximation of Generalized Processor Sharing with Interleaved Stratified Timer Wheels. *IEEE/ACM Trans. Netw.*, 18(3):708–721, 2010. URI: http://dx.doi.org/10.1109/TNET.2009.2033059, doi:10.1109/TNET.2009.2033059.

[51] F. Checconi, L. Rizzo, and P. Valente. QFQ: Efficient Packet Scheduling With Tight Guarantees. *Networking, IEEE/ACM Transactions on*, 21(3):802–816, June 2013. doi:10.1109/TNET.2012.2215881.

[52] J. Xu and R. J. Lipton. On Fundamental Tradeoffs Between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms. *SIGCOMM Comput. Commun. Rev.*, 32(4):279–292, 2002. URI: http://doi.acm.org/10.1145/964725.633052, doi:10.1145/964725.633052.

[53] L. Lenzini, E. Mingozzi, and G. Steay. Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers. *Networking, IEEE/ACM Transactions on*, 12(4):681–693, Aug 2004. doi:10.1109/TNET.2004.833131.

[54] P. Valente. Reducing the execution time of fair-queueing packet schedulers . *Computer Communications*, 47:16 – 33, 2014. URI: http://www.sciencedirect.com/science/article/pii/S0140366414001455, doi:http://dx.doi.org/10.1016/j.comcom.2014.04.009.

[55] A. Francini and F. M. Chiussi. Minimum-latency dual-leaky-bucket shapers for packet multiplexers: theory and implementation. In *Quality of Service, 2000. IWQOS. 2000 Eighth International Workshop on*, pages 19–28, 2000. doi:10.1109/IWQOS.2000.847935.

[56] D. Hang, H.-R. Shao, W. Zhu, and Y.-Q. Zhang. TD^{2}FQ: an integrated traffic scheduling and shaping scheme for DiffServ networks. In *High Performance Switching and Routing, 2001 IEEE Workshop on*, pages 78–82, 2001. doi:10.1109/HPSR.2001.923608.

[57] J. F. Lee, M. C. Chen, and Y. Sun. WF2Q-M: Worst-case fair weighted fair queueing with maximum rate control . *Computer Networks*, 51(6):1403 – 1420, 2007. URI: http://www.sciencedirect.com/science/article/pii/S1389128606002064, doi:http://dx.doi.org/10.1016/j.comnet.2006.07.013.

[58] J. F. Lee, M. C. Chen, and Y. Sun. WF2Q-M : a worst-case fair weighted fair queueing with maximum rate control. In *Global Telecommunications Conference, 2002. GLOBECOM 02. IEEE*, volume 2, pages 1576–1580 vol.2, Nov 2002. doi:10.1109/GLOCOM.2002.1188463.

[59] J. S. Hong. *Design of an Atm Shaping Multiplexer Algorithm and Architecture*. PhD thesis, Brooklyn, NY, USA, 1997. AAI9711852.

[60] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches. *Selected Areas in Communications, IEEE Journal on*, 15(5):938–950, Jun 1997. doi:10.1109/49.594854.

[61] H. J. Chao. Design of leaky bucket access control schemes in ATM networks. In *ICC 91 International Conference on Communications Conference Record*, pages 180–187 vol.1, Jun 1991. doi:10.1109/ICC.1991.162356.

[62] T. Moors, N. Clarke, and G. Mercankosk. Implementing traffic shaping. In *Local Computer Networks, 1994. Proceedings., 19th Conference on*, pages 307–314, 1994. doi:10.1109/LCN.1994.386589.

[63] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. A scalable architecture for fair leaky-bucket shaping. In *INFOCOM 97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 3, pages 1054–1062 vol.3, Apr 1997. doi:10.1109/INFCOM.1997.631123.

[64] J. Liebeherr and E. Yilmaz. Workconserving vs. non-workconserving packet scheduling: an issue revisited. In *Quality of Service, 1999. IWQoS 99. 1999 Seventh International Workshop on*, pages 248–256, 1999. doi:10.1109/IWQOS.1999.766500.

[65] R. L. Cruz. A calculus for network delay. I. Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, Jan 1991. doi:10.1109/18.61109.

[66] R. L. Cruz. A calculus for network delay. II. Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, Jan 1991. doi:10.1109/18.61110.

[67] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-node Case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, 1993. URI: http://dx.doi.org/10.1109/90.234856, doi:10.1109/90.234856.

[68] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Trans. Netw.*, 2(2):137–150, 1994. URI: http://dx.doi.org/10.1109/90.298432, doi:10.1109/90.298432.

[69] A. Kortebi, S. Oueslati, and J. Roberts. Implicit service differentiation using deficit round robin. *ITC19*, 2005.

[70] S. Jiwasurat, G. Kesidis, and D. J. Miller. Hierarchical shaped deficit round-robin scheduling. In *GLOBECOM 05. IEEE Global Telecommunications Conference, 2005.*, volume 2, pages 6 pp.–, Nov 2005. doi:10.1109/GLOCOM.2005.1577729.

[71] L. Lenzini, E. Mingozzi, and G. Stea. Bandwidth and Latency Analysis of Modified Deficit Round Robin Scheduling Algorithms. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, New York, NY, USA, 2006. ACM. URI: http://doi.acm.org/10.1145/1190095.1190147, doi:10.1145/1190095.1190147.

[72] L. Lenzini, E. Mingozzi, and G. Stea. Performance analysis of Modified Deficit Round Robin schedulers. *Journal of High Speed Networks*, 16(4):399–422, 2007. URI: http://iospress.metapress.com/content/p476t48201538820.

[73] Cisco Systems, Inc. Understand and Configure MDRR/WRED on the Cisco 12000 Series Internet Router, mar 2008. Retrieved 2016-12-21. URI: http://www.cisco.com/c/en/us/support/docs/routers/12000-series-routers/18841-mdrr-wred-18841.html#topic1.

[74] P. Southwick, D. Marschke, and H. Reynolds. *Junos Enterprise Routing: A Practical Guide to Junos Routing and Certification*. O'Reilly Media, 2011.

[75] D. Ros and M. Welzl. Less-than-Best-Effort Service: A Survey of End-to-End Approaches. *Communications Surveys Tutorials, IEEE*, 15(2):898–908, Second 2013. doi:10.1109/SURV.2012.060912.00176.

[76] M. Kühlewind, S. Neuner, and B. Trammell. On the state of ECN and TCP options on the Internet. In *Passive and active measurement*, pages 135–144. Springer, 2013.

[77] R. Craven, R. Beverly, and M. Allman. A Middlebox-cooperative TCP for a Non End-to-end Internet. *SIGCOMM Comput. Commun. Rev.*, 44(4):151–162, 2014. URI: http://doi.acm.org/10.1145/2740070.2626321, doi:10.1145/2740070.2626321.

[78] T. Stockhammer. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pages 133–144, New York, NY, USA, 2011. ACM. URI: http://doi.acm.org/10.1145/1943552.1943572, doi:10.1145/1943552.1943572.

[79] A. Aaron, Z. Li, M. Manohara, J. D. Cock, and D. Ronca. The Netflix Tech Blog: Per-Title Encode Optimization. Blog, dec 2015. Retrieved 2016-12-21. URI: http://techblog.netflix.com/2015/12/per-title-encode-optimization.html.

[80] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. L. Zhang. Unreeling netflix: Understanding and improving multi-CDN movie delivery. In *INFOCOM, 2012 Proceedings IEEE*, pages 1620–1628, March 2012. doi:10.1109/INFCOM.2012.6195531.

[81] D. Lee, B. E. Carpenter, and N. Brownlee. Observations of UDP to TCP Ratio and Port Numbers. In *Internet Monitoring and Protection (ICIMP), 2010 Fifth International Conference on*, pages 99–104, May 2010. doi:10.1109/ICIMP.2010.20.

[82] A. Feldmann and W. Willinger. Distilling the Internet's Application Mix from Packet-Sampled Traffic. In *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings*, volume 8995, page 179. Springer, 2015.

[83] Sandvine Inc. ULC. 2015 Global Internet Phenomena Report - Asia-Pacific and Europe. Technical report, September 2015. URI: https://www.sandvine.com/trends/global-internet-phenomena/.

[84] Sandvine Inc. ULC. 2016 Global Internet Phenomena Report - Africa, Asia-Pacific, and Middle East. Technical report, October 2016. URI: https://www.sandvine.com/trends/global-internet-phenomena/.

[85] Sandvine Inc. ULC. 2016 Global Internet Phenomena Report - Latin America and North America. Technical report, June 2016. URI: https://www.sandvine.com/trends/global-internet-phenomena/.

[86] A. Norberg. uTorrent Transport Protocol, August 2015. Retrieved 2016-12-21. URI: http://www.bittorrent.org/beps/bep_0029.html.

[87] Sandvine Inc. ULC. 2015 Global Internet Phenomena Report - Africa, Middle East and North America. Technical report, December 2015. URI: https://www.sandvine.com/trends/global-internet-phenomena/.

[88] L. Gharai. Rtp with tcp friendly rate control. Internet-Draft draft-ietf-avt-tfrc-profile-10, IETF Secretariat, July 2007. http://www.ietf.org/internet-drafts/draft-ietf-avt-tfrc-profile-10.txt. URI: http://www.ietf.org/internet-drafts/draft-ietf-avt-tfrc-profile-10.txt.

[89] I. working group. RTP Media Congestion Avoidance Techniques (rmcat). Retrieved 2016-06-16. URI: https://datatracker.ietf.org/wg/rmcat/charter/.

[90] C. Perkins. Rtp control protocol (rtcp) feedback for congestion control in interactive multimedia conferences. Internet-Draft draft-ietf-rmcat-rtp-cc-feedback-02, IETF Secretariat, October 2016. http://www.ietf.org/internet-drafts/draft-ietf-rmcat-rtp-cc-feedback-02.txt. URI: http://www.ietf.org/internet-drafts/draft-ietf-rmcat-rtp-cc-feedback-02.txt.

[91] Z. Sarker, C. Perkins, V. Singh, and M. Ramalho. Rtp control protocol (rtcp) feedback for congestion control. Internet-Draft draft-dt-rmcat-feedback-message-01, IETF Secretariat, October 2016. http://www.ietf.org/internet-drafts/draft-dt-rmcat-feedback-message-01.txt. URI: http://www.ietf.org/internet-drafts/draft-dt-rmcat-feedback-message-01.txt.

[92] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo. A google congestion control algorithm for real-time communication. Internet-Draft draft-ietf-rmcat-gcc-02, IETF Secretariat, July 2016. http://www.ietf.org/internet-drafts/draft-ietf-rmcat-gcc-02.txt. URI: http://www.ietf.org/internet-drafts/draft-ietf-rmcat-gcc-02.txt.

[93] ZDF. Multicast Adressen ZDF und ARD Programme, 2016. Retrieved 2017-01-18. URI: http://www.zdf.de/ZDF/zdfportal/blob/26516094/1/data.pdf.

[94] D.-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989. URI: http://dx.doi.org/10.1016/0169-7552(89)90019-6, doi:10.1016/0169-7552(89)90019-6.

[95] V. Jacobson. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols*, volume 18, pages 314–329, New York, NY, USA, 1988. ACM. URI: http://doi.acm.org/10.1145/52324.52356, doi:10.1145/52324.52356.

[96] The Institute of Electrical and Electronics Engineers, Inc. Ieee std 802.11-2012: Wireless LAN medium access control (MAC) and physical layer (PHY) specification. Technical Report 802.11-2012, IEEE 802.11, Piscataway, NJ, 2012.

[97] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end Arguments in System Design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984. URI: http://doi.acm.org/10.1145/357401.357402, doi:10.1145/357401.357402.

[98] S. Hemminger. tcp: remove appropriate byte count support, February 2013. Retrieved 2016-12-21. URI: http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=ca2eb5679f8ddfff60156af42595df44a315ef0.

[99] S. Alcock and R. Nelson. Application Flow Control in YouTube Video Streams. *SIGCOMM Comput. Commun. Rev.*, 41(2):24–30, 2011. URI: http://doi.acm.org/10.1145/1971162.1971166, doi:10.1145/1971162.1971166.

[100] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate Limiting YouTube Video Streaming. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 191–196, Boston, MA, 2012. USENIX. URI: https://www.usenix.org/conference/atc12/technical-sessions/presentation/ghobadi.

[101] Y. C. Eric Dumazet. Tso, fair queuing, pacing: three's a charm. Slides at TCPM Meetingm at 88. IETF Meeting, November 2013. Retrieved 2016-12-21. URI: http://www.ietf.org/proceedings/88/slides/slides-88-tcpm-9.pdf.

[102] B. Briscoe. Flow Rate Fairness: Dismantling a Religion. *SIGCOMM Comput. Commun. Rev.*, 37(2):63–74, 2007. URI: http://doi.acm.org/10.1145/1232919.1232926, doi:10.1145/1232919.1232926.

[103] N. Dukkipati and N. McKeown. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, 2006. URI: http://doi.acm.org/10.1145/1111322.1111336, doi:10.1145/1111322.1111336.

[104] M. Welzl. *Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems)*. John Wiley & Sons, 2005.

[105] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock. Host-to-Host Congestion Control for TCP. *IEEE Communications Surveys Tutorials*, 12(3):304–342, Third 2010. doi:10.1109/SURV.2010.042710.00114.

[106] W. Lautenschläger. A Deterministic Bandwidth Sharing Model. *CoRR*, abs/1404.4173, 2014. URI: http://arxiv.org/abs/1404.4173.

[107] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. *SIGCOMM Comput. Commun. Rev.*, 22(2):9–16, 1992. URI: http://doi.acm.org/10.1145/141800.141801, doi:10.1145/141800.141801.

[108] D. Rossi. Open Source Scavenging Transport Protocols, 2013. URI: http://www.telecom-paristech.fr/~drossi/paper/rossi13scavenging.pdf.

[109] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Täht. Interaction or Interference: Can AQM and Low Priority Congestion Control Successfully Collaborate? In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, pages 25–26, New York, NY, USA, 2012. ACM. URI: http://doi.acm.org/10.1145/2413247.2413263, doi:10.1145/2413247.2413263.

[110] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht. Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 411–416, April 2013. doi:10.1109/INFCOMW.2013.6562885.

[111] A. Kuzmanovic and E. W. Knightly. TCP-LP: a distributed algorithm for low priority data transfer. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1691–1701 vol.3, March 2003. doi:10.1109/INFCOM.2003.1209192.

[112] A. Kuzmanovic and E. Knightly. TCP-LP: low-priority service via end-point congestion control. *Networking, IEEE/ACM Transactions on*, 14(4):739–752, Aug 2006. doi:10.1109/TNET.2006.879702.

[113] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A Mechanism for Background Transfers. *SIGOPS Oper. Syst. Rev.*, 36(SI):329–343, 2002. URI: http://doi.acm.org/10.1145/844128.844159, doi:10.1145/844128.844159.

[114] A. Cooper and S. Dawkins. Congestion Control For Interactive Real-Time Communication - The IAB and IAB/IRTF Workshop, July 2012. Retrieved 2016-12-21. URI: https://www.iab.org/wp-content/IAB-uploads/2012/10/Congestion-Control-Workshop-Minutes.pdf.

[115] Microsoft. About BITS. Retrieved 2016-12-21. URI: https://msdn.microsoft.com/en-us/library/aa362708(v=vs.85).aspx.

[116] C. Caini and R. Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, 2004. URI: http://dx.doi.org/10.1002/sat.799, doi:10.1002/sat.799.

[117] S. Bensley, L. Eggert, D. Thaler, P. Balasubramanian, and G. Judd. Datacenter tcp (dctcp): Tcp congestion control for datacenters. Internet-Draft draft-ietf-tcpm-dctcp-02, IETF Secretariat, July 2016. http://www.ietf.org/internet-drafts/draft-ietf-tcpm-dctcp-02.txt. URI: http://www.ietf.org/internet-drafts/draft-ietf-tcpm-dctcp-02.txt.

[118] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, New York, NY, USA, 2010. ACM. URI: http://doi.acm.org/10.1145/1851182.1851192, doi:10.1145/1851182.1851192.

[119] V. Jacobson. Modified tcp congestion avoidance algorithm. Email to the end2end mailing list at the Information Sciences Institute, April 1990. Retrieved 2016-12-21. URI: ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[120] L. S. Brakmo and L. L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, Oct 1995. doi:10.1109/49.464716.

[121] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu. TCP Congestion Avoidance Algorithm Identification. *IEEE/ACM Transactions on Networking*, 22(4):1311–1324, Aug 2014. doi:10.1109/TNET.2013.2278271.

[122] T. Hoff. 7 years of youtube scalability lessons in 30 minutes. Retrieved 2016-12-21. URI: http://highscalability.com/blog/2012/3/26/7-years-of-youtube-scalability-lessons-in-30-minutes.html.

[123] C. Do. Seattle conference on scalability: Youtube scalability. Youtube Video. Retrieved 2017-01-18. URI: https://www.youtube.com/watch?v=w5WVu624fY8.

[124] B. Gregg. The netflix tech blog - linux performance analysis in 60,000 milliseconds, November 2015. Retrieved 2017-01-18. URI: http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html.

[125] Apple Inc. Sources of the latest os x at the time of writing, version 10.11.6, file *tcp_ledbat.c*. Retrieved 2016-12-21. URI: https://opensource.apple.com/source/xnu/xnu-3248.60.10/bsd/netinet/tcp_ledbat.c.

[126] M. Mathis, J. Semke, and J. Mahdavi. The rate-halving algorithm for tcp congestion control. Internet-Draft draft-mathis-tcp-ratehalving-00, IETF Secretariat, August 1999. https://tools.ietf.org/html/draft-mathis-tcp-ratehalving-00.txt. URI: https://tools.ietf.org/html/draft-mathis-tcp-ratehalving-00.txt.

[127] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008. URI: http://doi.acm.org/10.1145/1400097.1400105, doi:10.1145/1400097.1400105.

[128] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, March 2004. doi:10.1109/INFCOM.2004.1354672.

[129] D. Leith and R. Shorten. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet*, volume 2004, 2004.

[130] D. Leith. H-tcp: Tcp congestion control for high bandwidth-delay product paths. Internet-Draft draft-leith-tcp-htcp-06, IETF Secretariat, April 2008. http://www.ietf.org/internet-drafts/draft-leith-tcp-htcp-06.txt. URI: http://www.ietf.org/internet-drafts/draft-leith-tcp-htcp-06.txt.

[131] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. LEDBAT: The New BitTorrent Congestion Control Protocol. In *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pages 1–6, Aug 2010. doi:10.1109/ICCCN.2010.5560080.

[132] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti. Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol . *Computer Networks*, 57(8):1838 – 1852, 2013. URI: http://www.sciencedirect.com/science/article/pii/S1389128613000765, doi:http://dx.doi.org/10.1016/j.comnet.2013.02.020.

[133] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Täht. Fighting the bufferbloat: On the coexistence of and low priority congestion control . *Computer Networks*, 65(0):255 – 267, 2014. URI: http://www.sciencedirect.com/science/article/pii/S1389128614000188, doi:http://dx.doi.org/10.1016/j.bjp.2014.01.009.

[134] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa. A hands-on assessment of transport protocols with lower than best effort priority. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 8–15, Oct 2010. doi:10.1109/LCN.2010.5735831.

[135] A. J. Abu and S. Gordon. Impact of Delay Variability on LEDBAT Performance. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pages 708–715, March 2011. doi:10.1109/AINA.2011.98.

[136] N. Kuhn, O. Mehani, A. Sathiaseelan, and E. Lochin. Less-than-Best-Effort Capacity Sharing over High BDP Networks with LEDBAT. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–5, Sept 2013. doi:10.1109/VTCFall.2013.6692266.

[137] S. Q. V. Trang, N. Kuhn, E. Lochin, C. Baudoin, E. Dubois, and P. Gelard. On the existence of optimal LEDBAT parameters. In *2014 IEEE International Conference on Communications (ICC)*, pages 1216–1221, June 2014. doi:10.1109/ICC.2014.6883487.

[138] BitTorrent Inc. uTorrent Transport Protocol library. Retrieved 2017-01-18. URI: https://github.com/bittorrent/libutp.

[139] M. Granatiero. Vergleichende Bewertung von Lower-Than-Best-Effort Congestion Control Algorithmen. Bachelor Thesis, Universität Stuttgart, 2015.

[140] J. Schneider, J. Wagner, R. Winter, and H. J. Kolbe. Out of my way - evaluating Low Extra Delay Background Transport in an ADSL access network. In *Teletraffic Congress (ITC), 2010 22nd International*, pages 1–8, Sept 2010. doi:10.1109/ITC.2010.5608714.

[141] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, April 2006. doi:10.1109/INFOCOM.2006.188.

[142] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound tcp: A new tcp congestion control for high-speed and long distance networks. Internet-Draft draft-sridharan-tcpm-ctcp-02, IETF Secretariat, November 2008. http://www.ietf.org/internet-drafts/draft-sridharan-tcpm-ctcp-02.txt. URI: http://www.ietf.org/internet-drafts/draft-sridharan-tcpm-ctcp-02.txt.

[143] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4):281–292, 2004. URI: http://doi.acm.org/10.1145/1030194.1015499, doi:10.1145/1030194.1015499.

[144] C. Villamizar and C. Song. High Performance TCP in ANSNET. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, 1994. URI: http://doi.acm.org/10.1145/205511.205520, doi:10.1145/205511.205520.

[145] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[146] M. Allman. Comments on Bufferbloat. *SIGCOMM Comput. Commun. Rev.*, 43(1):30–37, 2012. URI: http://doi.acm.org/10.1145/2427036.2427041, doi:10.1145/2427036.2427041.

[147] C. Chirichella and D. Rossi. To the Moon and back: Are Internet bufferbloat delays really that large? In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 417–422, April 2013. doi:10.1109/INFCOMW.2013.6562886.

[148] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *Commun. ACM*, 55(1):57–65, 2012. URI: http://doi.acm.org/10.1145/2063176.2063196, doi:10.1145/2063176.2063196.

[149] Federal Communications Commission. Open internet. Website. Retrieved on 2017-01-18. URI: https://www.fcc.gov/general/open-internet.

[150] Sandvine Inc. ULC. Network Congestion Management: Considerations and Techniques. Webdownload, 2015. Retrieved 2016-12-21.

[151] R. B. Briscoe. *Re-feedback: Freedom with Accountability for Causing Congestion in a Connectionless Internetwork*. PhD thesis, 2009. URI: URL:url{http://www.cs.ucl.ac.uk/staff/B.Briscoe/pubs.html#refb-dis}.

[152] S. Dörner. Bewertung der Leistungsfähigkeit von ConEx-Policing. Master's thesis, Universität Stuttgart, 2013.

[153] working group. Congestion Exposure (conex). Retrieved 2017-01-18. URI: https://datatracker.ietf.org/wg/conex/documents/.

[154] M. Kühlewind, D. P. Wagner, J. M. R. Espinosa, and B. Briscoe. Using data center TCP (DCTCP) in the Internet. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 583–588, Dec 2014. doi:10.1109/GLOCOMW.2014.7063495.

[155] D. P. Wagner. Congestion Policing Queues - A new approach to managing bandwidth sharing at bottlenecks. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 206–211, Nov 2014. doi:10.1109/CNSM.2014.7014160.

[156] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet packet size distributions: Some observations. *USC/Information Sciences Institute, Tech. Rep. ISI-TR-2007-643*, 2007.

[157] S. S. Kanhere and H. Sethu. Fair, efficient and low-latency packet scheduling using nested deficit round robin. In *High Performance Switching and Routing, 2001 IEEE Workshop on*, pages 6–10, 2001. doi:10.1109/HPSR.2001.923594.

[158] H. Kocher and M. Lang. An object-oriented library for simulation of complex hierarchical systems. *SIMULATION SERIES*, 26:145–145, 1994.

[159] Institute of Communication Networks and Computer Engineering (IKR). Institute of Communication Networks and Computer Engineering (IKR) - IKR Simulation Library - Getting the Libraries. Retrieved 2017-01-18. URI: http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/Download/.

[160] T. Werthmann, M. Kaschub, M. Kühlewind, S. Scholz, and D. Wagner. VMSimInt: A Network Simulation Tool Supporting Integration of Arbitrary Kernels and Applications. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 56–65, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). URI: http://dx.doi.org/10.4108/icst.simutools.2014.254623, doi:10.4108/icst.simutools.2014.254623.

[161] L. Torvalds et al. Linux kernel release 4.4.6. Web site, March 2016. Retrieved 2017-01-18. URI: https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.6.tar.xz.

[162] J. Sing and B. Soh. TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links. In *Fourth IEEE International Symposium on Network Computing and Applications*, pages 73–82, July 2005. doi:10.1109/NCA.2005.52.

[163] D. A. Hayes and G. Armitage. Revisiting Congestion Control using Delay Gradients. In *Proc. of IFIP Networking*, pages 328–341, Valencia, Spain, 2011. Springer. URI: http://dx.doi.org/10.1007/978-3-642-20798-3_25.

[164] A. Kuzmanovic and E. W. Knightly. TCP-LP: Low-priority Service via End-point Congestion Control. *IEEE/ACM Trans. Netw.*, 14(4):739–752, 2006. URI: http://dx.doi.org/10.1109/TNET.2006.879702, doi:10.1109/TNET.2006.879702.

[165] C. P. Fu and S. C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, 21(2):216–228, Feb 2003. doi:10.1109/JSAC.2002.807336.

[166] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 287–297, New York, NY, USA, 2001. ACM. URI: http://doi.acm.org/10.1145/381677.381704, doi:10.1145/381677.381704.

[167] A. C. F. Vacirca, A. Baiocchi. YeAH-TCP: yet another highspeed TCP. Proceedings of Fifth International Workshop on Protocols for FAST Long-Distance Networks (PFLDnet 2007), Marina Del Rey, CA, USA, Feb 2007.

[168] A. Currid. TCP Offload to the Rescue. *Queue*, 2(3):58–65, 2004. URI: http://doi.acm.org/10.1145/1005062.1005069, doi:10.1145/1005062.1005069.

[169] J. C. Mogul. TCP Offload is a Dumb Idea Whose Time Has Come. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association. URI: http://dl.acm.org/citation.cfm?id=1251054.1251059.

[170] A. P. Foong, T. R. Huff, H. H. Hum, J. R. Patwardhan, and G. J. Regnier. TCP performance re-visited. In *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on*, pages 70–79, March 2003. doi:10.1109/ISPASS.2003.1190234.

[171] R. Prasad, M. Jain, and C. Dovrolis. *Effects of Interrupt Coalescence on Network Measurements*, pages 247–256. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. URI: http://dx.doi.org/10.1007/978-3-540-24668-8_25, doi:10.1007/978-3-540-24668-8_25.

[172] K. Kant. TCP offload performance for front-end servers. In *Global Telecommunications Conference, 2003. GLOBECOM 03. IEEE*, volume 6, pages 3242–3247 vol.6, Dec 2003. doi:10.1109/GLOCOM.2003.1258835.

[173] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance characterization of a 10-Gigabit Ethernet TOE. In *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*, pages 58–63, Aug 2005. doi:10.1109/CONECT.2005.30.

[174] M. Zec, M. Mikuc, and M. Žagar. Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput. In *in Proceedings of 10-th SoftCOM*, 2002.

[175] R. Raghavendra and E. M. Belding. Characterizing high-bandwidth real-time video traffic in residential broadband networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010 Proceedings of the 8th International Symposium on*, pages 597–602, May 2010.

[176] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network Characteristics of Video Streaming Traffic. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, pages 25:1–25:12, New York, NY, USA, 2011. ACM. URI: http://doi.acm.org/10.1145/2079296.2079321, doi:10.1145/2079296.2079321.

[177] B. A. Mah. An empirical model of HTTP network traffic. In *INFOCOM 97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 2, pages 592–600 vol.2, Apr 1997. doi:10.1109/INFCOM.1997.644510.

[178] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 151–160, New York, NY, USA, 1998. ACM. URI: http://doi.acm.org/10.1145/277851.277897, doi:10.1145/277851.277897.

[179] H.-K. Choi and J. O. Limb. A behavioral model of Web traffic. In *Network Protocols, 1999. (ICNP 99) Proceedings. Seventh International Conference on*, pages 327–334, Oct 1999. doi:10.1109/ICNP.1999.801961.

[180] F. Hernandez-Campos, K. Jeffay, and F. D. Smith. Tracking the evolution of Web traffic: 1995-2003. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.*, pages 16–25, Oct 2003. doi:10.1109/MASCOT.2003.1240638.

[181] R. Pries, Z. Magyari, and P. Tran-Gia. An HTTP web traffic model based on the top one million visited web pages. In *Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on*, pages 133–139, June 2012. doi:10.1109/NGI.2012.6252145.

[182] J. J. Lee and M. Gupta. A New Traffic Model for Current User Web Browsing Behavior. Technical report, 2007. URI: http://blogs.intel.com/wp-content/mt-content/com/research/HTTP%20Traffic%20Model_v1%201%20white%20paper.pdf.

[183] R. Jain, D.-M. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, September 1984.

[184] A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 4th edition, 2007.

[DOCSIS] Cable Television Laboratories, Inc. DOCSIS 3.1 Physical Layer Specification cm-sp-phyv3.1-i09-160602. Specification 3.1, Cable Television Laboratories, Inc., June 2016.

[802.1q] IEEE. IEEE Std 802.1Q-2014 bridges and bridged networks. Specification 802.1Q, IEEE, March 2014.

[G.114] ITU-T. One-way transmission time. Rec. G.114, ITU-T, May 2003.

[G.9701] ITU-T. Fast access to subscriber terminals (G.fast). Rec. G.971, ITU-T, December 2014.

[G.987] ITU-T. 10-gigabit-capable passive optical network (XG-PON) systems: Definitions, abbreviations and acronyms. Rec. G.987, ITU-T, June 2012.

[G.987.2] ITU-T. 10-gigabit-capable passive optical networks (XG-PON): Physical media dependent (PMD) layer specification. Rec. G.987.2, ITU-T, February 2016.

[G.989.2] ITU-T. 40-Gigabit-capable passive optical networks 2 (NG-PON2): Physical media dependent (PMD) layer specification. Rec. G.989.2, ITU-T, December 2014.

[G.992.1] ITU-T. Asymmetric digital subscriber line (ADSL) transceivers. Rec. G.992.1, ITU-T, July 1999.

[G.992.3] ITU-T. Asymmetric digital subscriber line transceivers 2 (ADSL2). Rec. G.992.3, ITU-T, January 2005.

[G.992.5] ITU-T. Asymmetric Digital Subscriber Line (ADSL) transceivers - Extended bandwidth ADSL2 (ADSL2+). Rec. G.992.5, ITU-T, January 2005.

[G.993.1] ITU-T. Very high speed digital subscriber line transceivers. Rec. G.993.1, ITU-T, June 2004.

[G.993.2] ITU-T. Very high speed digital subscriber line transceivers 2 (VDSL2). Rec. G.993.1, ITU-T, June 2004.

[X.200] ITU-T. Information technology - Open Systems Interconnection - Basic Reference Model: The basic model. Rec. X.200, ITU-T, July 1994.

[RFC793] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528. URI: `http://www.ietf.org/rfc/rfc793.txt`.

[RFC826] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (INTERNET STANDARD), November 1982. Updated by RFCs 5227, 5494. URI: `http://www.ietf.org/rfc/rfc826.txt`.

[RFC896] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896 (Historic), January 1984. Obsoleted by RFC 7805. URI: `http://www.ietf.org/rfc/rfc896.txt`.

[RFC970] J. Nagle. On Packet Switches With Infinite Storage. RFC 970, December 1985. URI: `http://www.ietf.org/rfc/rfc970.txt`.

[RFC1122] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (INTERNET STANDARD), October 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. URI: http://www.ietf.org/rfc/rfc1122.txt.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996. URI: http://www.ietf.org/rfc/rfc2018.txt.

[RFC2177] B. Leiba. IMAP4 IDLE command. RFC 2177 (Proposed Standard), June 1997. URI: http://www.ietf.org/rfc/rfc2177.txt.

[RFC2581] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390. URI: http://www.ietf.org/rfc/rfc2581.txt.

[RFC3168] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040. URI: http://www.ietf.org/rfc/rfc3168.txt.

[RFC3465] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental), February 2003. URI: http://www.ietf.org/rfc/rfc3465.txt.

[RFC3550] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD), July 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164. URI: http://www.ietf.org/rfc/rfc3550.txt.

[RFC3782] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard), April 2004. Obsoleted by RFC 6582. URI: http://www.ietf.org/rfc/rfc3782.txt.

[RFC4654] J. Widmer and M. Handley. TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification. RFC 4654 (Experimental), August 2006. URI: http://www.ietf.org/rfc/rfc4654.txt.

[RFC4861] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007. Updated by RFCs 5942, 6980, 7048, 7527, 7559. URI: http://www.ietf.org/rfc/rfc4861.txt.

[RFC5166] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), March 2008. URI: http://www.ietf.org/rfc/rfc5166.txt.

[RFC5348] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard), September 2008. URI: http://www.ietf.org/rfc/rfc5348.txt.

[RFC5681] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009. URI: http://www.ietf.org/rfc/rfc5681.txt.

[RFC5760] J. Ott, J. Chesterfield, and E. Schooler. RTP Control Protocol (RTCP) Extensions for Single-Source Multicast Sessions with Unicast Feedback. RFC 5760 (Proposed Standard), February 2010. Updated by RFC 6128. URI: http://www.ietf.org/rfc/rfc5760.txt.

[RFC5851] S. Ooghe, N. Voigt, M. Platnic, T. Haag, and S. Wadhwa. Framework and Requirements for an Access Node Control Mechanism in Broadband Multi-Service Networks. RFC 5851 (Informational), May 2010. URI: http://www.ietf.org/rfc/rfc5851.txt.

[RFC6057] C. Bastian, T. Klieber, J. Livingood, J. Mills, and R. Woundy. Comcast's Protocol-Agnostic Congestion Management System. RFC 6057 (Informational), December 2010. URI: http://www.ietf.org/rfc/rfc6057.txt.

[RFC6093] F. Gont and A. Yourtchenko. On the Implementation of the TCP Urgent Mechanism. RFC 6093 (Proposed Standard), January 2011. URI: http://www.ietf.org/rfc/rfc6093.txt.

[RFC6128] A. Begen. RTP Control Protocol (RTCP) Port for Source-Specific Multicast (SSM) Sessions. RFC 6128 (Proposed Standard), February 2011. URI: http://www.ietf.org/rfc/rfc6128.txt.

[RFC6298] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298 (Proposed Standard), June 2011. URI: http://www.ietf.org/rfc/rfc6298.txt.

[RFC6320] S. Wadhwa, J. Moisand, T. Haag, N. Voigt, and T. Taylor. Protocol for Access Node Control Mechanism in Broadband Networks. RFC 6320 (Proposed Standard), October 2011. Updated by RFC 7256. URI: http://www.ietf.org/rfc/rfc6320.txt.

[RFC6333] A. Durand, R. Droms, J. Woodyatt, and Y. Lee. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333 (Proposed Standard), August 2011. Updated by RFC 7335. URI: http://www.ietf.org/rfc/rfc6333.txt.

[RFC6528] F. Gont and S. Bellovin. Defending against Sequence Number Attacks. RFC 6528 (Proposed Standard), February 2012. URI: http://www.ietf.org/rfc/rfc6528.txt.

[RFC6582] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Proposed Standard), April 2012. URI: http://www.ietf.org/rfc/rfc6582.txt.

[RFC6675] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, and Y. Nishida. A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP. RFC 6675 (Proposed Standard), August 2012. URI: http://www.ietf.org/rfc/rfc6675.txt.

[RFC6789] B. Briscoe, R. Woundy, and A. Cooper. Congestion Exposure (ConEx) Concepts and Use Cases. RFC 6789 (Informational), December 2012. URI: http://www.ietf.org/rfc/rfc6789.txt.

[RFC6817] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low Extra Delay Background Transport (LEDBAT). RFC 6817 (Experimental), December 2012. URI: http://www.ietf.org/rfc/rfc6817.txt.

[RFC6928] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Window. RFC 6928 (Experimental), April 2013. URI: http://www.ietf.org/rfc/rfc6928.txt.

[RFC6937] M. Mathis, N. Dukkipati, and Y. Cheng. Proportional Rate Reduction for TCP. RFC 6937 (Experimental), May 2013. URI: http://www.ietf.org/rfc/rfc6937.txt.

[RFC7713] M. Mathis and B. Briscoe. Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements. RFC 7713 (Informational), December 2015. URI: http://www.ietf.org/rfc/rfc7713.txt.

[RFC7778] D. Kutscher, F. Mir, R. Winter, S. Krishnan, Y. Zhang, and C. Bernardos. Mobile Communication Congestion Exposure Scenario. RFC 7778 (Informational), March 2016. URI: http://www.ietf.org/rfc/rfc7778.txt.

[RFC7786] M. Kuehlewind and R. Scheffenegger. TCP Modifications for Congestion Exposure (ConEx). RFC 7786 (Experimental), May 2016. URI: http://www.ietf.org/rfc/rfc7786.txt.

[RFC7837] S. Krishnan, M. Kuehlewind, B. Briscoe, and C. Ralli. IPv6 Destination Option for Congestion Exposure (ConEx). RFC 7837 (Experimental), May 2016. URI: http://www.ietf.org/rfc/rfc7837.txt.

[TR-59] DSL Forum. DSL Evolution - Architecture Requirements for the Support of QoS-Enabled IP Services. Technical Report TR-59, DSL Forum, September 2003.

[TR-101] DSL Forum. Migration to Ethernet-Based DSL Aggregation. Technical Report TR-101, DSL Forum, April 2006.

[TR-134] The Broadband Forum. Broadband Policy Control Framework (BPCF). Technical Report TR-134, Broadband Forum, July 2012.

A Acknowledgments

First of all, my thanks go to Prof. Dr.-Ing. Andreas Kirstädter for all the research opportunities and chances that he provided at the Institute of Communication Networks and Computer Engineering (IKR) and the very open and supporting guidance during the creation of this thesis. I also thank him and Prof. Dr. Michael Menth for the assessment of this written thesis.

My special thanks go to Ulrich Gemkow. He created an atmosphere at the IKR that was always defined by as much as possible freedom in research and high standards in all aspects of research and teaching. Uli always had his own way of fostering the best of all members of the institute. I also thank for Uli's feedback, this thesis benefited a lot from it.

Many thanks go to my fellow researchers at the IKR. We had so many fruitful and enlightening discussions, but also so many events that helped to free my mind. I'd like to thank in particular Mirja Kühlewind, Sebastian Meier and Frank Feller for listening and discussing. I'd like to thank all of my fellow mortarboard makers: I learned much about microcontrollers and on how distributed projects with hard deadlines can be completed timely and successfully. I always enjoyed these times with my IKR colleagues.

I'm also very grateful for the valuable input and help in different aspects from the people who reviewed parts of the draft of this thesis, namely Marc Barisch, Uwe Bauknecht, Dorothea Dombrowski, Frank Feller, Ester Gallardo, Ulrich Gemkow, Sebastian Meier, Sebastian Scholz and Jürgen Wagner.

I greatly appreciate the cooperation and exchange with fellow researchers, especially Mirja Kühlewind, Bob Briscoe and Wolfram Lautenschläger. It was a pleasure and again and again enlightening to work and discuss with you. Actually, Bob's input and ideas regarding ConEx were the starting point of my interest and fascination for congestion management and thus for this work.

The work with bachelor and master students during their thesis projects helped me exploring a wider range in the field of congestion control research and their often divergent perspective helped me deepening my understanding in this field. My thanks go to all the students who by this supported the creation of this work.

Another group that definitely deserves my thanks are my former colleagues at Fraunhofer FOKUS: Karl Jonas, the head of our little research group in the Birlinghoven outpost, who always motivated me to go further and actually motivated me to approach a doctorate. I am also very thankful for the many opportunities Karl opened up. I thank Jens Mödeker and Mathias

Kretschmer for all the good discussions, their patience with me and for the many things I learned from them.

I also thank the many research colleagues in the projects I had the opportunity to take part in: Daidalos, NetQoS, SelfNet, G-Lab, ETICS, SASER. The role of these colleagues changed over time: At the beginning, research colleagues mostly inspired me and expanded my view on networks, most prominently in Daidalos II. In later projects, the fellow researchers rather were valuable discussion partners to me and touchstones for new ideas. All of them helped me to make my way to this point.

My deepest thanks go to my family and my girlfriend Thea. My parents always supported me and also motivated me to strive for more. They provided the basis for all this. Finally, I sincerely thank the one person who supported me in so many ways during the completion of this thesis as well as during the preparation for the oral exam: my wonderful girlfriend Thea.