Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

New Methods for Efficient Data Transport in Future Networks

Ph.D. Dissertation

Author:
Zoltán Móczár

Supervisor:
Dr. Sándor Molnár
Associate Professor

Budapest, Hungary
2015

Abstract

Over the past decades, the congestion control algorithm of the *Transmission Control Protocol (TCP)* has played a key role in providing reliable communication between Internet hosts. Since the deployment of TCP, networks have undergone a significant change due to the emerging technologies, the diversity of applications and the growing number of users. This process has led to a myriad of TCP variants intended to achieve better performance under various network conditions. However, the long research history of TCP suggests that its incremental development and continuous optimization for specific target environments cannot keep up with the current trends in networking, hence there is an urgent need for novel data transport methods based on fundamentally different principles.

This dissertation addresses the challenging task of building an Internet architecture without the functionality of congestion control while still ensuring reliable end-to-end communication. As a possible solution, a new data transmission paradigm built upon digital fountain codes is presented and investigated. The main component of our concept is a transport mechanism called *Digital Fountain based Communication Protocol (DFCP)*, which runs on the top of a network architecture where fairness is managed at the routers instead of endpoints. In the first part of the dissertation, the design and operating principles are introduced together with the discussion of the potential benefits. The second part presents a comprehensive performance analysis of DFCP and TCP carried out in a multi-platform evaluation framework using testbed measurements and packet-level simulations. Since today's networks are highly variable, the third part is devoted to the characterization of the two transfer paradigms under dynamic traffic conditions. Finally, a bandwidth estimation method is proposed for mobile networks, which can estimate the available bandwidth by exploiting the user-generated downlink network traffic with negligible additional load. Overall, our results point out that the digital fountain based approach is an efficient and promising way of reliable data transfer well-suited to a broad range of future applications.

Acknowledgments

First of all, I would like to express my deepest gratitude to my supervisor, *Sándor Molnár*, for his guidance, support and encouragement throughout my PhD studies that made it possible to become a researcher. He taught me the way of thinking and how to present scientific results, as well as shared a lot of knowledge and personal experience.

I wish to thank *Balázs Sonkoly* for his valuable advice, ideas and for the assistance in coping with technical issues. Thanks are also due to *Szilárd Solymos* for the thorough development, and to all of my undergraduate students who contributed to the results. In addition, I am indebted to *Péter Megyesi* for the common work we have done in the framework of our joint research projects.

I am also thankful to the colleagues at Ericsson TrafficLab including *László Kovács*, *András Veres*, *Tamás Borsos*, *Sándor Rácz*, *Géza Szabó* and *Attila Mihály* for research cooperations, which gave me the opportunity to deal with industry-relevant topics. My special thanks go to *Szilveszter Nádas* for the exciting and useful discussions on the field of congestion control and resource management.

My work was done in the High Speed Networks Laboratory (HSNLab) hosted by the Department of Telecommunications and Media Informatics (TMIT) at the Budapest University of Technology and Economics (BME). I say thanks to the head of our laboratory, *Attila Vidács*, for the financial support of my research that enabled me to attend several leading international conferences.

Last but not least, I would like to thank *my whole family* for their continuous support during my student years. I am especially grateful to *my wife, Réka*, her unconditional love and patience provided me a stable background to carry out my research.

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership Project
ACCF	Adaptive Congestion Control Framework
ACK	Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
ARQ	Automatic Repeat reQuest
ATCP	Ad-hoc TCP
BDP	Bandwidth-Delay Product
BIC	Binary Increase Congestion control
BSD	Berkeley Software Distribution
BTC	Bulk Transfer Capacity
CCA	Congestion Control based Architecture
CDF	Cumulative Distribution Function
ConEx	Congestion Exposure
CTCP	Compound TCP
DCTCP	Data Center TCP
DFA	Digital Fountain based Architecture
DFCP	Digital Fountain based Communication Protocol
DRR	Deficit Round-Robin
DVB-H	Digital Video Broadcasting — Handheld
ECN	Explicit Congestion Notification
EDGE	Enhanced Data rates for GSM Evolution
FBP	Fountain Based Protocol
FCT	Flow Completion Time
FEC	Forward Error Correction
FIFO	First In First Out

FMTCP	Fountain code-based Multipath TCP
GENI	Global Environment for Network Innovations
GPRS	General Packet Radio Service
HSDPA	High-Speed Downlink Packet Access
HSTCP	HighSpeed TCP
HTTP	HyperText Transfer Protocol
IAT	Inter-Arrival Time
IETF	Internet Engineering Task Force
IP	Internet Protocol
LDPC	Low-Density Parity-Check
LEDBAT	Low Extra Delay Background Transport
LT	Luby Transform
MBMS	Multimedia Broadcast Multicast Service
MIMD	Multiplicative Increase Multiplicative Decrease
MPTCP	MultiPath TCP
MTU	Maximum Transmission Unit
NSC	Network Simulation Cradle
PCC	Performance-oriented Congestion Control
PGM	Probe Gap Model
PRM	Probe Rate Model
QoE	Quality of Experience
QST	Queue Saturation Time
QUIC	Quick UDP Internet Connections
RCP	Rate Control Protocol
RFC	Request for Comments
RTT	Round-Trip Time
SACK	Selective Acknowledgment
SCTP	Stream Control Transmission Protocol
SDN	Software-Defined Networking
SDT	Software-Defined Transport
SFQ	Stochastic Fair Queuing
SSL	Secure Sockets Layer
STCP	Scalable TCP

TCL	Tool Command Language
TCP	Transmission Control Protocol
TCP-LP	TCP Low Priority
TCP/NC	TCP with Network Coding
TLS	Transport Layer Security
UDP	User Datagram Protocol
UDT	UDP-based Data Transport
UMTS	Universal Mobile Telecommunications System
WFQ	Weighted Fair Queuing
WLAN	Wireless Local Area Network
XCP	eXplicit Control Protocol
XOR	eXclusive OR

Chapter 1

Introduction

1.1 Motivation and Objectives

From the very beginning of the Internet, traffic congestion has been recognized as an undesirable phenomenon that must be avoided in order to maintain stable operation. Congestion occurs when the aggregate demand for a resource exceeds its capacity, which typically leads to significant performance degradation in communication networks. As a solution, the *Transmission Control Protocol (TCP)* [1] was introduced in 1981 and it defined a set of mechanisms to prevent such adverse events by adjusting the transmission rate based on various observations. Closed-loop congestion control performed by TCP has proven to be a successful approach, but several versions have been developed over the past decades to satisfy the ever-changing requirements of heterogeneous network environments [2]. However, in the recent years it became apparent that the continuous refinement of TCP cannot follow the incredibly fast evolution of technologies and applications, as well as the increasing user demands.

It is clear that emerging paradigms like cloud computing and software-defined networking, and what is more, the upcoming era of 5G mobile networks and Internet of Things will require much more efficient *transport methods governed by fundamentally different principles*. Taking into account these trends, it is natural to ask if congestion control is indispensable to ensure reliable communication. While the research community is urged to find the answer, only a few papers elaborate on this challenging issue. For example, the authors of [3] argue that it may not be necessary to keep the network uncongested to yield good performance, and a greedy transport protocol has the potential to outperform TCP. In 2007, a research plan [4] published by the organization of GENI (Global Environment for Network Innovations) recommended the omission of TCP's congestion

control mechanism and suggested to use error correction techniques instead so as to cope with packet loss. The validity of this approach is supported by a recent study [5] claiming that congestion collapse does not happen in many cases even if no congestion control is applied at the network endpoints. To investigate whether the Internet can work efficiently without the key functionality of TCP, extensive research needs to be conducted.

In this dissertation, we take an important step towards the understanding of *data transmission in the absence of congestion control*. We aimed to carry out a clean-slate research and to design the concept of reliable transport from scratch. Our main contribution is the investigation of a novel digital fountain based communication paradigm, which consists of a transport protocol and an underlying network architecture where congestion control is completely omitted. First, we present a comprehensive performance analysis of the new paradigm in a multi-platform evaluation framework and make a comparison with the traditional TCP-based solution of current Internet. We explore the fundamental features of the different mechanisms with a special focus on their characteristics under dynamic traffic conditions. Furthermore, we deal with available bandwidth estimation when there is no information about the network in advance, and propose an algorithm capable of providing accurate results. We believe that our findings greatly promote the research on alternative data transfer methods as we can deliver the message that communication without the need of controlling the congestion is possible and merits further analysis from several aspects.

1.2 Research Methodology

The *Digital Fountain based Communication Protocol (DFCP)*, presented in Chapter 3, has been implemented in the Linux kernel. To deeply investigate our proposal, measurements were conducted both in a *simulation framework* and in *real test networks*. As a *simulation tool*, the Network Simulation Cradle (NSC) [6] has been integrated into the widely known ns-2 packet-level simulator [7], and extended in C++ to properly handle the kernel implementation of DFCP. Testbed measurements were performed in different *laboratory network configurations* and in a remote *network emulation environment* called Emulab [8]. In order to get sound results, these three platforms have been extensively cross-validated, which is unique in a sense regarding the literature of transport protocols. The available bandwidth estimation algorithm, introduced in Chapter 6, was evaluated on packet traces gathered from a 3G mobile network.

1.3 Structure of the Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we review the evolution of transport protocols since the early days of the Internet discussing the main pitfalls the researchers faced with during the last decades. We present different types of congestion control algorithms highlighting their advantages and drawbacks, as well as introduce some recent alternative proposals for data transfer. We close this chapter with the main consequences of the long history of transport protocols and shed light on the motivation for future research by concluding state-of-the-art papers.

Chapter 3 presents a novel network communication paradigm built upon a digital fountain based transport mechanism abbreviated as DFCP, which completely omits the functionality of congestion control. We introduce the operating principles and the potential benefits of the new approach together with a discussion of the main protocol design aspects. In addition, we describe our carefully cross-validated evaluation environment consisting of real testbeds and a simulation framework used to investigate the nature of digital fountain based communication. At the end of Chapter 3, the fundamental properties of DFCP are revealed.

In Chapter 4, we elaborate on the performance analysis of two completely different data transfer paradigms, namely fountain coding and congestion control. We aim to explore the main benefits of DFCP over the traditional TCP-based approach in a variety of network environments. The comprehensive performance evaluation of transport mechanisms covers a wide range of capabilities including the maximum achievable transmission rate in lossy and high-latency networks, the buffer space demand and usage, the transfer efficiency of short-lived and long-lived flows, the bandwidth sharing among concurrent traffic flows, as well as network utilization and scalability.

Chapter 5 is devoted to the characterization of transport mechanisms under continuously changing network conditions. We carry out well-designed experiments to understand the dynamic behavior of the digital fountain and congestion control based paradigms, respectively. First, we investigate the convergence speed of these mechanisms to the steady-state operating phase, and examine how stable the performance they can provide in the long run. Moreover, the property of responsiveness is thoroughly analyzed in order to reveal the ability to handle abrupt change of network parameters.

In Chapter 6, we deal with available bandwidth estimation recognized as an important task in the context of data communication. After an overview of the related work, we introduce our bandwidth estimation algorithm especially tailored for mobile networks,

which is based on the idea of busy period detection. We demonstrate the operability of our algorithm on a packet trace gathered in a cellular network by using a realistic traffic emulator.

Finally, Chapter 7 summarizes the contributions of the dissertation and draws our main conclusions. The potential future applications of the new results together with a brief discussion are also given. Furthermore, we sketch some open issues and possible research directions.

Chapter 2

The Evolution of Transport Protocols

In the current Internet, the Transmission Control Protocol (TCP) carries the vast majority of network traffic. The history of TCP dates back to 1981 when the official protocol specification was published by the IETF in RFC 793 [1]. Over the past three decades, a significant research effort has been devoted to TCP in order to meet the requirements of evolving communication networks. This process has resulted in countless TCP versions aimed to provide high performance in various environments [2]. Although TCP determined the mainstream of the research on transport protocols, in the last years many alternative proposals have also been published to serve as the basis of reliable data communication. In this chapter, we give an overview of the most widely known protocols including the main TCP variants and other approaches, as well. Finally, we review some interesting recent work intended to highlight the challenges of handling today's heterogeneous set of congestion control mechanisms and the architectural deficiency of TCP, which strongly motivate future research on fundamentally different paradigms.

2.1 Transmission Control Protocol

TCP is a connection-oriented transport protocol that provides reliable data transfer in end-to-end communication. It means that lost packets are retransmitted, and therefore, each sent packet will eventually be delivered to the destination. One of the most important features of TCP is its *congestion control* mechanism, which is used to avoid congestion collapse [9] by determining the proper sending rate and to achieve high performance. To this end, TCP maintains a congestion window (*cwnd*) that controls the number of outstanding unacknowledged packets in the network. An important aspect in the context of congestion control protocols is how they can share the available bandwidth among

Table 2.1. The evolution of TCP variants

Version	Congestion indicator		Target environment			New features
	Loss	*Delay*	*Wired*	*Wireless*	*High-speed*	
TCP Tahoe [1] ➠ *1988*	×		×			slow-start, congestion avoidance and fast retransmit
TCP Reno [25] ➠ *1990*	×		×			fast recovery to mitigate the impact of packet losses
TCP Vegas [31] ➠ *1995*		×	×			bottleneck buffer utilization as a congestion feedback
TCP NewReno [27] ➠ *1999*	×		×			fast recovery, resistance to multiple losses
Freeze-TCP [18] ➠ *2000*	×			×		considering radio signal quality in mobile networks
TCP-Peach [19] ➠ *2001*	×			×		sudden start and rapid recovery for satellite networks
TCP Westwood [29] ➠ *2001*	×			×		estimation of the available bandwidth
ATCP [20] ➠ *2001*	×			×		detection of route changes in ad-hoc networks
TCP Nice [21] ➠ *2002*		×	×			delay threshold as a secondary congestion indicator
Scalable TCP [12] ➠ *2003*	×		×		×	MIMD congestion avoidance algorithm
TCP-LP [22] ➠ *2003*	×		×			early congestion detection to react sooner than TCP
HighSpeed TCP [13] ➠ *2003*	×		×		×	AIMD mechanism as the function of *cwnd*
FAST TCP [14] ➠ *2003*		×	×		×	updating *cwnd* based on different equations
BIC TCP [28] ➠ *2004*	×		×		×	binary search to find the proper *cwnd*
Compound TCP [32] ➠ *2005*	×	×	×		×	calculation of *cwnd* using loss and delay components
TCP-Illinois [33] ➠ *2006*	×	×	×		×	AIMD as the function of the queuing delay
TCP Cubic [15] ➠ *2008*	×		×		×	control of *cwnd* by applying a cubic function
LEDBAT [23] ➠ *2012*		×	×			congestion control for low-priority traffic

competing flows, also known as fairness property. Fairness can be interpreted between the same and different TCP versions (intra- and inter-protocol), as well as on various time scales (transient and steady-state) [10].

TCP variants can be classified based on the type of congestion indication and the target environment as shown in Table 2.1. Most congestion control methods use packet loss information to detect congestion also known as *loss-based* TCPs. In the case of these algorithms, packet loss is interpreted as the sign of a full network buffer from which the last incoming packet was dropped, hence the transmission rate should be reduced. Another group of congestion control mechanisms react to the increase observed in the round-trip

time (RTT) of packets due to the building up of queues. This approach, often referred to as *delay-based* TCP, has the ability to detect congestion early rather than merely waiting until the network gets overutilized and packets are lost. In addition, *hybrid* solutions have also been proposed, which combine the beneficial properties of loss-based and delay-based algorithms.

During the years, the fast development of networks motivated researchers to optimize TCP for certain environments and purposes by modifying the traditional congestion control mechanism. Since standard TCP versions (like TCP Tahoe and Reno) failed to obtain full utilization in networks with high-bandwidth links, new algorithms have been introduced to improve the performance in such conditions [11]. The most relevant *high-speed* TCP versions include Scalable TCP [12], HighSpeed TCP [13], FAST TCP [14] and TCP Cubic [15]. On the other hand, as TCP was primarily designed for wired networks, emerging wireless communication induced a considerable research work to develop TCP versions, which can provide better performance in different kinds of *wireless* networks [16, 17]. The performance issues experienced in such environments stem from the unique characteristics of wireless links and the packet loss model used by TCP. The problems manifest in many applications as degradation of throughput, inefficiency in network resource utilization and excessive interruption of data transmissions. Modification of standard TCP for wireless communication has been an active research area in recent years, and many schemes have been proposed for various environments such as cellular (e.g. Freeze-TCP [18]), satellite (e.g. TCP-Peach [19]) and ad-hoc networks (e.g. ATCP [20]). In real networks a traffic mix consists of hundreds or thousands of flows generated by diverse applications and services. In order to treat low-priority traffic (e.g. background transfers like automatic software updates and data backups) differently from high-priority traffic, *low-priority* congestion control methods have been introduced. These protocols, such as TCP Nice [21], TCP-LP [22] and LEDBAT [23], respond to congestion earlier than standard TCP yielding bandwidth to concurrent TCP flows with higher priority.

2.1.1 Loss-based Versions

One of the earliest approaches to handle congestion was introduced in TCP Tahoe [9], which was also served as the first practical implementation of these control schemes in the BSD operating system. The proposal is based on the original TCP specification [1] and introduces new algorithms called *slow-start (SS)* and *congestion avoidance (CA)*, as illustrated in Figure 2.1. These mechanisms allow the sender to detect available network

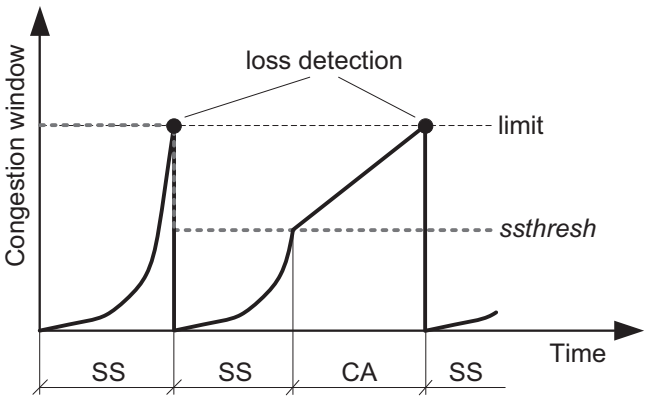


Figure 2.1. Slow-start and congestion avoidance algorithms

resources and adjust the transmission rate accordingly. In the slow-start phase the window growth follows an exponential function. If a packet loss is detected due to the total utilization of network resources, the congestion window (*cwnd*) is reset to the initial value. The congestion avoidance algorithm is aimed at improving TCP effectiveness in networks with limited resources. In this operating phase, the congestion window increases by one only if all data packets have been successfully delivered during the last round-trip time, and it is merely halved when a packet loss is detected (the mechanism is abbreviated as AIMD: additive increase multiplicative decrease [24]). To switch between the two algorithms, a threshold parameter (*ssthresh*) is introduced. This threshold determines the maximum size of the congestion window in the slow-start phase, and each packet loss adjusts its value to half of the current window size. The congestion window itself is always reset to a minimum value upon loss detection. Until the size of the congestion window is lower than *ssthresh*, the slow-start algorithm is used. Once the window becomes greater than the threshold, the protocol enters the congestion avoidance phase. However, the main problem with Tahoe is that it reduces the congestion window too aggressively upon loss detection.

TCP Reno [25] tackles the deficiency of Tahoe by applying a novel method referred to as *fast recovery (FR)* algorithm, which keeps the congestion window size constant until the network is recovered from the loss event (see Figure 2.2). In the case of Reno, a lost packet is detected and retransmitted if triple duplicate acknowledgments are received or a timeout occurs at the sender. This mechanism makes TCP Reno effective to recover from a single packet loss, but it still suffers from performance degradation when multiple packets are dropped from a window of data. To overcome this limitation, a selective acknowledgment (SACK) option has been proposed in [26]. Later, in order to improve

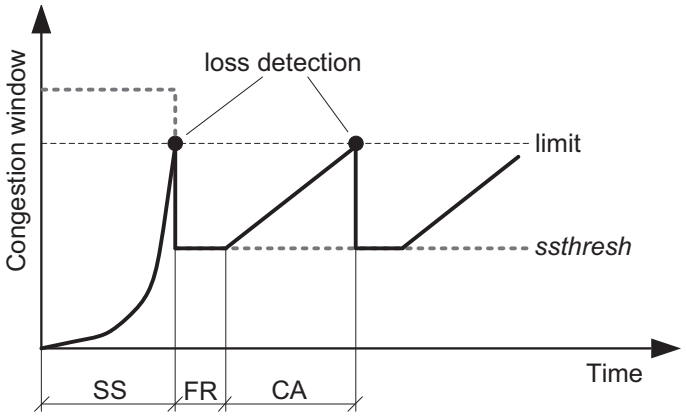


Figure 2.2. The congestion window dynamics of TCP Reno

the performance of TCP Reno when a burst of packets is lost, TCP NewReno [27] was developed in 1999. NewReno modifies Reno's fast recovery algorithm making it possible to recover without a retransmission timeout by resending one packet per each round-trip time until all of the lost packets from the window have been retransmitted.

TCP Cubic [15], being an enhanced version of its predecessor, BIC TCP [28], is one of the most widely used TCP versions today since it serves as the default congestion control algorithm of Linux operating systems. BIC (Binary Increase Congestion control) was originally designed to solve the well-known RTT unfairness problem by combining two schemes called *additive increase* and *binary search*. TCP Cubic simplifies the window control of BIC and it applies a *cubic function* in terms of the elapsed time from the previous packet loss event, which provides good stability and scalability. Furthermore, it keeps the window growth rate independent of RTT making the protocol TCP-friendly along both short and long RTT paths. According to the TCP-friendliness principle, any congestion control scheme has to achieve equal long-term throughput, or in other words,

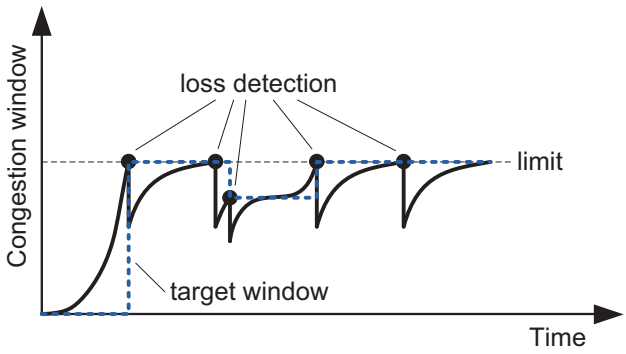


Figure 2.3. The window growth function of TCP Cubic

has to consume equal bandwidth of a bottleneck link as TCP if they are operated in the same network environment. The congestion window dynamics of TCP Cubic (see Figure 2.3) is governed by the following equation:

$$w = C(t - K)^3 + w_{max}. \tag{2.1}$$

In the above formula, C is a scaling factor, t denotes the elapsed time since the last window reduction and w_{max} gives the target window size. The value of parameter K can be determined as follows:

$$K = \sqrt[3]{w_{max}\frac{\beta}{C}} \tag{2.2}$$

where β is a multiplicative decrease factor applied for window reduction at the time of a congestion event. When data transmission starts, the target window size is unknown and discovered using the right branch of the cubic function. In this phase, the growth speed is slower than the exponential discovery of the slow-start algorithm, and at later stages the congestion window gently approaches the target window.

Beside the congestion control algorithms described above, many other solutions have been worked out to improve the performance of standard TCP. One of the main issues is that it takes a long time to make a full recovery from packet loss for high-bandwidth, long-distance connections, because the congestion window builds up very slowly. In order to cope with this limitation, HighSpeed TCP (HSTCP) [13] was proposed, which can achieve better performance on *high-capacity links* by modifying the congestion control algorithm for use with large congestion windows. Scalable TCP (STCP) [12] applies a multiplicative increase multiplicative decrease (MIMD) algorithm to obtain performance improvement in high-speed networks and it can also guarantee the *scalability* of the protocol. TCP Westwood [29] is a sender-side modification of the congestion control mechanism that improves the performance of TCP Reno both in wired and wireless networks. The main problem is that TCP Reno equally reacts to random and congestion losses, thus cannot distinguish between them. In fact, TCP Westwood shows moderate sensitivity to random errors, therefore the improvement is the most significant in *wireless networks with lossy links*. MultiPath TCP (MPTCP) [30] is a recent approach for enabling the *simultaneous use of multiple IP addresses* or interfaces by a modification of TCP that presents a regular TCP interface to applications, while spreading data across several subflows.

2.1.2 Delay-based Versions

As a pioneer of delay-based TCPs, TCP Vegas [31] measures the difference (δ) between the expected and actual throughput based on round-trip delays. If δ is less than a lower threshold denoted by α, Vegas assumes that the path is not congested and increases the sending rate. If δ is larger than an upper threshold denoted by β, it is regarded as the indication of congestion, hence Vegas reduces the transmission rate. The expected throughput is calculated as the division of *cwnd* by the minimum RTT.

FAST TCP [14] is a congestion avoidance algorithm especially targeted for long-distance, high-latency links. FAST determines the current congestion window size using the round-trip delay as a primary congestion indicator. The algorithm first estimates the queuing delay based on RTTs over a network path, and if the delay falls below a threshold, it increases the window aggressively. If it gets closer to the threshold, the algorithm slowly reduces the increasing rate.

2.1.3 Hybrid Solutions

Compound TCP (CTCP) [32], implemented in several Microsoft Windows operating systems, is a synergy of delay-based and loss-based approaches extending the standard TCP Reno congestion avoidance algorithm by a scalable, delay-based component. CTCP exploits the information about both packet loss and delay to control the transmission rate. The delay-based component can rapidly increase the sending rate when the network path is underutilized, but ease if the bottleneck queue becomes full. This mechanism provides good scalability in terms of bandwidth, and a reasonably fair behavior.

TCP-Illinois [33] uses packet loss information to determine whether the congestion window size should be increased or decreased, and measures the queuing delay to determine the amount of increment or decrement. This hybrid solution makes it possible to obtain high throughput and fair resource allocation while being compatible with standard TCP.

2.2 Alternative Proposals

Beyond the Transmission Control Protocol, several approaches have also been suggested for reliable data transport in communication networks. Some of these protocols are partially based on the concept of TCP, or use similar mechanisms.

Stream Control Transmission Protocol (SCTP) [34] is a reliable transport protocol that provides stable, ordered delivery of data between two endpoints by using congestion control like TCP and also preserves data message boundaries like UDP (User Datagram Protocol). However, unlike TCP and UDP, SCTP offers additional services such as multi-homing, multi-streaming, security and authentication.

eXplicit Control Protocol (XCP) [35] uses direct congestion notification instead of the indirect congestion indicators such as packet loss or delay. XCP delivers the highest possible application performance over a broad range of network infrastructures including high-speed and high-delay links where TCP performs poorly. It also introduces a novel way for separating the efficiency and fairness policies of congestion control, enabling routers to quickly make use of the available bandwidth while conservatively managing the allocation of the available bandwidth to competing flows. XCP carries the per-flow congestion state in the packet header allowing the sender to request a desired throughput for its transmission, and XCP-capable routers inform the senders about the degree of congestion at the bottleneck.

Internet traffic has a complex characteristics investigated in many papers in the last decade. Recent studies showed that most flows are small carrying only several kilobytes of data and short lasting less than a few seconds [36]. Rate Control Protocol (RCP) [37] is a congestion control algorithm designed to significantly speed up the download of short-lived flows generated by typical applications. For example, a mid-size flow contains 1000 packets and TCP makes them last nearly 10 times longer than it would be necessary. RCP enables flows to finish close to the minimum possible, leading to a notable improvement for web users and distributed file systems.

Quick UDP Internet Connections (QUIC) [38] is a new approach for data transfer announced by Google in 2013. QUIC has been integrated into Google Chrome for evaluation purposes and it is currently under active development. The protocol supports a set of multiplexed connections over UDP, and was designed to provide security protection equivalent to TLS/SSL, along with reduced connection and transport latency. It also implements and applies a bandwidth estimation algorithm in each direction in order to avoid network congestion. QUIC's main goal is to optimize the performance of connection-oriented web applications and services by the reduction of connectivity overhead to zero RTT. Until now, no comprehensive evaluation has been carried out on the performance of QUIC, but a recent study showed that it can reduce the overall page retrieval time with respect to HTTP in the case of a channel without induced random losses [39].

2.3 Present and Future

The history of transport protocols has been dominated by the continuous refinement and optimization of TCP for various environments over the past decades. Countless TCP variants have been proposed to achieve performance gain under specific conditions, but most of them have never been used in real networks. Moreover, this long development process has led to an unmanageable set of congestion control algorithms with several interoperability and fairness issues.

An Adaptive Congestion Control Framework (ACCF) is presented in [40] to cover a wide range of network conditions by automatically switching among different loss-based and delay-based congestion control mechanisms depending on the current network state. The operation of ACCF was investigated through extensive experiments conducted both in simulation and testbed environments. The authors found that ACCF can significantly improve performance compared to other state-of-the-art algorithms in terms of throughput, fairness and TCP-friendliness. The study presented in [41] examines what aspects should be taken into account in the design phase of a distributed transport mechanism, including the network parameters (e.g. link delay and capacity), the topology, the degree of multiplexing and the aggressiveness of contending endpoints. The authors carry out a quantitative analysis by using an automated protocol-design tool to approximate the best possible congestion control scheme given imperfect prior knowledge about the network. Their surprising results point out that, in many cases, the performance of machine-generated optimal protocols can attain and even surpass the performance of human-designed congestion control algorithms.

As for now, due to the incredibly rapid growth of communication networks and services, it is clear that the current practice of making yet another TCP version to tackle the upcoming challenges becomes more and more hopeless, and many researchers suggest to find fundamentally different solutions for reliable data transport in future Internet.

The authors of [42] believe that the root cause of problems is an architectural deficiency of TCP. They claim that the reason why TCP variants have suffered from poor performance for decades is the hardwiring of packet-level events to control responses. In addition, the authors do not think that TCP can achieve consistent high performance if this control policy remains unchanged. To cope with this issue, they propose Performance-oriented Congestion Control (PCC), a new congestion control architecture in which each sender continuously observes the connections between its actions and empirically experienced performance, enabling it to consistently adopt actions that result in high perfor-

mance. As a control action, PCC chooses a certain sending rate then calculates a utility value depending on measured performance metrics such as throughput, loss rate and latency. The utility function can be customized for various data transmission objectives, but typically it maximizes the overall throughput and minimizes the packet loss rate. According to the results, PCC increases its sending rate if this action leads to higher utility, otherwise the sending rate will be decreased. The paper shows that across many real-world environments, PCC can significantly outperform TCP while converging to a stable and fair equilibrium.

There is no doubt of the need for investigating new approaches, and in the following chapter we present a novel data transfer paradigm, which is able to eliminate several shortcomings of TCP and offers a promising alternative for future communication networks.

Chapter 3

A Digital Fountain Based Network Communication Paradigm

Over the years, the issues of TCP motivated researchers to find alternative ways for data transfer beside the traditional congestion control based approach. In 2007, an organization called GENI [4] published a research plan, in which the authors recommend the omission of the congestion control mechanism and suggest to use efficient erasure coding techniques to cope with network congestion. Since then, the questions related to this idea have been investigated only in a few papers. Raghavan and Snoeren argue in [3] that it may not be necessary to keep the network uncongested to achieve good performance and fairness. They introduce the concept of decongestion control and presume that a protocol relying upon greedy, high-speed transmission has the potential to perform better than TCP. Bonald et al. studied the consequences of operating a network without congestion control [5], and concluded that it does not inevitably lead to congestion collapse as believed earlier. In this chapter, we present and describe a novel data transfer paradigm for future Internet, which applies a fundamentally different principle compared to that of TCP by completely omitting congestion control from the transport layer.

3.1 Background

3.1.1 Digital Fountain Codes

Fountain codes, also known as *rateless codes*, are a class of erasure codes with the property that a potentially limitless sequence of encoded symbols (n) can be generated from a given set of source symbols (k) such that the original source symbols can ideally be recovered from any subset of the encoded symbols of size equal to, or only slightly larger than the

number of source symbols [43]. As illustrated in Figure 3.1, in the case of the digital fountain principle it does not matter what is lost if enough is received. In contrast to traditional erasure codes, rateless codes do not have a fixed coding rate, and this coding rate tends to zero as the length of the encoded message tends to infinity (i.e. $\frac{k}{n} \to 0$ if $n \to \infty$).

Figure 3.1. The digital fountain principle

The development of fountain codes over the past decades can be seen in Figure 3.2 with the official publication date of each proposal. Historically, *Tornado codes* [44] were the first generation of erasure codes intended to approximate the principle outlined above. The basic structure of a Tornado code consists of layered bipartite graphs with left and right nodes corresponding to the message and check symbols, respectively. A check symbol is generated by computing the XOR (eXclusive OR) of the values of its neighboring message nodes. The encoding process works in the following way. Let assume that we have the bipartite graphs $B_1, B_2, \ldots, B_l$ and C. The graph B_1 has k message symbols as input and produces βk check symbols where $0 < \beta < 1$. These output symbols serve as the input symbols of the next layer B_2, hence $\beta^2 k$ new check symbols are generated. This step is repeated for each layer B_i $(i = 1, 2, \ldots, l)$, and finally, the result is concatenated with a conventional error-correcting code C. The decoding of the message can be accomplished by using a simple belief propagation algorithm. Given the value of the check symbol and all but one of the message symbols on which it depends, the missing symbol is set to be the XOR of the check symbol and its known neighboring message symbols. The decoding process terminates successfully when all of the original symbols are recovered.

As Tornado codes were proven to be impractical due to the requirement for a cascade of graphs, they were quickly replaced by irregular *Luby Transform (LT) codes* [45], which have much simpler structure and equal or even better performance. To generate an en-

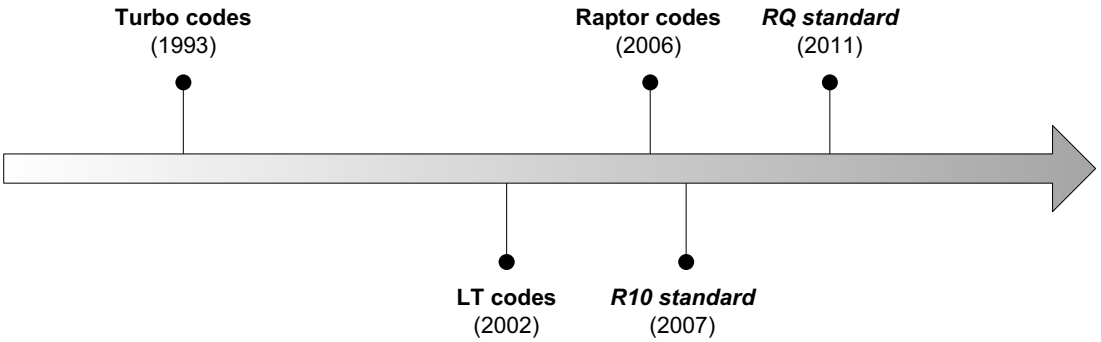


Figure 3.2. The evolution of digital fountain codes

coded symbol, a degree d is chosen randomly according to a given distribution $\Omega(d)$ where $1 \leq d \leq k$ and $\sum_{d=1}^{k} \Omega(d) = 1$. The degree d determines the number of message symbols involved in the generation of an encoded symbol, which are then chosen at random and XORed with each other. This encoding process exhibit a fountain-like property, because as many encoded symbols as desired can be generated efficiently. In order to ensure proper decoding, the random number generator must be initialized with the same predefined seed at both the encoder and decoder sides. After that, the original message symbols can be recovered using a similar procedure as in the case of Tornado codes. The efficiency of LT codes is highly determined by the choice of the degree distribution. According to the theoretical analysis presented in [45], the maximum performance can be obtained if the ideal soliton distribution is used, namely

$$\Omega(d) = \begin{cases} \frac{1}{k} & d = 1 \\ \frac{1}{d(d-1)} & d = 2, 3, \ldots, k \end{cases}. \tag{3.1}$$

The average number of neighbors of each encoded symbol, and therefore, the expected number of XOR operations is proportional to

$$\sum_{d=2}^{k} \frac{d}{d(d-1)} = \sum_{d=2}^{k} \frac{1}{d-1} \approx \ln(k). \tag{3.2}$$

This degree distribution guarantees that the belief propagation algorithm can recover a source block of k symbols from slightly more than k received encoded symbols with high probability. However, (3.1) works poorly in practice, and Luby proposes to use a so-called robust soliton distribution instead [45], which is designed for asymptotic optimality and

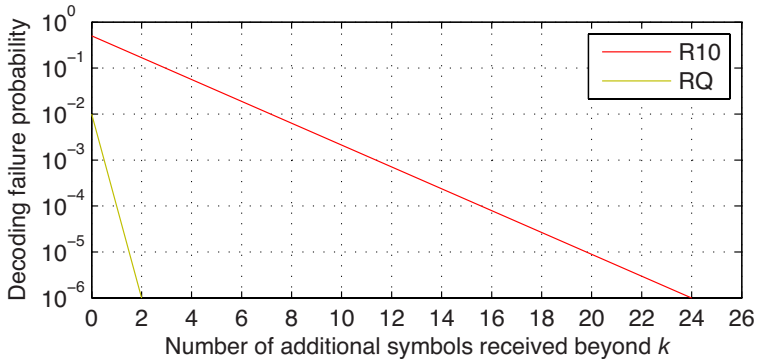


Figure 3.3. The decoding failure probability of different Raptor codes

covers various applications. The main drawback of LT codes is that they are unable to provide low complexity encoding and decoding operations.

In these days, *Raptor codes* [46] are the most efficient ones in the family of fountain codes. Their most significant improvement over LT codes is the reduction of the average degree of encoding symbols to a constant, which is achieved by an additional precoding phase (the operation is discussed in Section 3.3). As a result, Raptor codes offer linear time encoding and decoding complexity as they require only a small number of XOR operations proportional to k for each generated symbol. The first version of Raptor codes (also known as *R10*) is specified in [47], and it has also been adopted into multiple standards in the area of broadcast file delivery and streaming services (e.g. 3GPP MBMS, DVB-H) [48, 49]. Currently, the most flexible and powerful variant of Raptor codes is *RaptorQ* (or simply *RQ*) [50], which supports larger source block sizes (up to 56403 symbols) and provides better coding efficiency than R10. The decoding failure probability is shown in Figure 3.3 for different number of encoding symbols collected by the decoder beyond the original message length k. The illustration reveals the outstanding recovery properties of Raptor codes. For example, if two additional symbols are available, the chance of decoding failure for RaptorQ is only one in a million.

3.1.2 Advanced Error Correction in Data Transport

In recent times, many research works have focused on the application of erasure codes in data transport. A theoretical fountain based protocol (FBP) was investigated in [51]. The authors showed that a Nash equilibrium can be reached in a network with FBP-based hosts resulting in a performance similar to the case when each host uses TCP. Kumar et al. proposed a transport protocol for wireless networks using fountain codes [52] and

analyzed its performance by a Markovian stochastic model. They demonstrated through packet-level simulations that their protocol may perform better or worse than TCP depending on the redundancy parameter, the number of nodes in a WLAN cell and the wireless channel conditions. The authors of [53] designed a new TCP version on the basis of rateless erasure codes to enhance its operation in lossy environments. According to their results, such modification of TCP has proven to be effective in case of high packet loss rate. Y. Cui and his colleagues proposed FMTCP (Fountain code-based Multipath TCP) in [54], which exploits the advantage of the fountain coding scheme to avoid the performance degradation caused by frequent retransmissions applied in MPTCP. The authors introduced an algorithm to flexibly allocate encoded symbols to different subflows based on the expected packet arrival time over different paths. In another proposal called TCP/NC, network coding is incorporated into TCP with only minor changes to the protocol stack [55]. According to this method, the source transmits random linear combinations of packets currently found in the congestion window. Coding essentially masks losses from the congestion control algorithm and allows TCP/NC to react smoothly to them providing an effective solution for lossy environments like wireless networks. As a fundamentally new approach, this dissertation introduces a reliable, digital fountain based transport mechanism where no congestion control is employed at the endpoints.

3.2 Networking without Congestion Control

In this section, we envision a network architecture built upon digital fountain based communication and highlight the benefits of the approach with some potential future applications. The main component of the architecture is a novel data transport mechanism, which provides reliable transmission by efficient erasure coding and inherently makes it possible to get rid of congestion control and all related tasks at the transport layer.

3.2.1 Operating Principles

The novel data transfer method uses efficient erasure coding schemes to recover lost packets instead of traditional retransmissions. This approach enables endpoints to transmit at the *maximum possible rate*, thus the network can easily be driven to a state with heavily congested, fully utilized links. In our transport protocol, we propose to use Raptor codes [46] as a forward error correction (FEC) mechanism to cope with packet losses, which is an extension of LT codes with linear time encoding and decoding complexity.

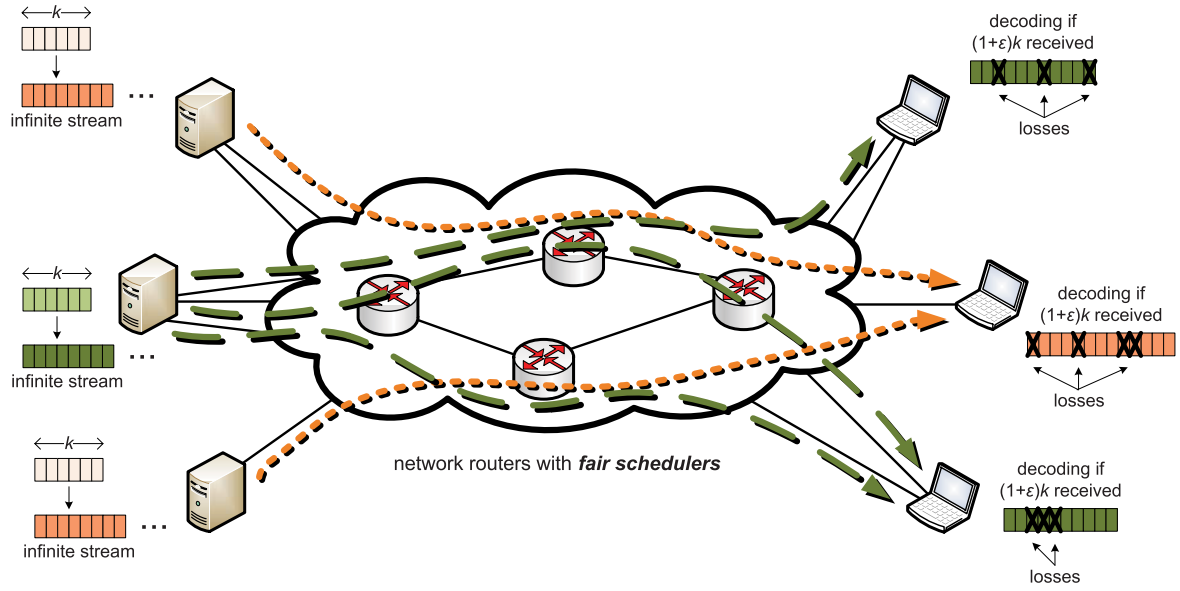


Figure 3.4. The digital fountain based communication architecture

The suggested network architecture relying on *digital fountain based error correction* is shown in Figure 3.4. We have multiple senders communicating with the corresponding receivers by producing a potentially infinite stream of encoded symbols from the original message of size k. Each received packet at the destination host increases the probability of successful decoding, and once any subset of size $\lceil (1+\varepsilon)k \rceil$ encoded symbols arrive to the receiver, decoding can be performed successfully with high probability (here $\epsilon > 0$ denotes the amount of redundancy added to the original message). One of the most important issues that must be resolved by this novel network architecture is fairness. More exactly, mechanisms have to be provided in order to give a solution to the share allocation problem among competing traffic flows sending at different rates. To this end, we suggest the use of *fair schedulers* in the network nodes since several implementations approximating the ideal fair scheduling, such as Deficit Round-Robin (DRR) [56], are available and can be configured easily in network routers. If equal bandwidth sharing is provided by the inner nodes then it becomes possible to decouple fairness control from the transport layer protocol. The feasibility of this approach is supported by the scalability of per-flow fair queuing, as its complexity does not increase with link capacity [57, 58].

3.2.2 Rate Control

Greedy transmission at the maximum rate can easily lead to an operational state when a huge number of packets are steadily sent via some parts of the network, but reaching

a bottleneck, they are dropped. This unnecessary wasting of available bandwidth, also known as *dead packet phenomenon* [59], can be avoided in several ways.

The sender could perform *passive or active measurements* on the currently available bandwidth along its network path like in the case of UDT (UDP-based Data Transport) [60]. Available bandwidth estimation has received considerable attention in the last decades due to its key role in many areas of networking such as transport layer protocols, admission control, network management and multimedia streaming (for a literature overview, please see Chapter 6). Different estimation techniques work with different overhead, speed and estimation error. In fact, it is almost impossible to obtain very precise estimation results because of the fast and dynamic change of traffic conditions, however, the proposed transfer mechanism does not require high accuracy. One of the key principles of our concept is to operate the network in the overloaded regime, which makes it possible to fully utilize the available resources. Of course, this approach leads to a considerable amount of packet loss at the network nodes, but from the user's point of view goodput-based QoE (Quality of Experience) metrics will only slightly be affected even in case of high congestion levels. Although the consequences of shifting the operation to the overloaded regime is a relevant aspect to be considered by the network operator, a rough estimate of the bottleneck bandwidth is still sufficient to reduce the packet drop rate at the buffers, and to keep it in an acceptable range. The measurement frequency depends on the applied algorithm, but it is practical to perform estimation such that it can roughly follow the network dynamics without causing significant overhead.

Another possibility to adjust the source rate properly is using a mechanism capable of *providing feedback about network congestion*, for instance, as XCP does. One of the most widely known solutions is called ECN (Explicit Congestion Notification) [61], which allows to signal congestion by marking packets instead of dropping them from the buffer. The re-ECN [62] protocol extends the ECN mechanism in order to inform the routers along a path about the estimated level of congestion. Today, network elements at any layer may signal congestion to the receiver by dropping packets or by ECN markings, and the receiver passes this information back to the sender in a transport layer feedback. ConEx (Congestion Exposure) [63] is a recent proposal, currently being standardized by IETF, that enables the sender to relay the congestion information back into the network in-band at the IP layer, such that the total amount of congestion from each element on the path is revealed to all nodes, which can be used to provide input for traffic management. SDN-based (Software-Defined Networking) mechanisms can also help to cope

with this issue where the network domains have dedicated central controllers with central knowledge regarding the domains, hence they could provide information on the available bandwidth to senders. For example, OpenTCP [64] is an SDN-based framework, which takes advantage of the global network view available at the controller to make faster and more accurate congestion control decisions.

3.2.3 Potential Benefits

The proposed networking paradigm offers a suitable framework for a wide range of applications and use-cases. For example, our scheme supports not only unicast type traffic but inherently provides efficient solution for *multicast and broadcast services*. The more challenging n-to-1 and n-to-n communication patterns including multiple servers can also be realized in a straightforward manner due to the beneficial properties of the fountain coding based approach, as it does not matter which part of the message is received, and it can be guaranteed that each received block provides extra information. In addition, our transport mechanism enables *multipath communication*, which has received a great interest in the recent years because of its potential to achieve higher network resiliency and load balancing targets. Another possible application area is *data centers* since the solution fits very well to the high utilization requirement of such environments. Moreover, our transport protocol is insensitive to packet loss and delay in contrast to TCP making it a good candidate for *wireless networks*. The deployment in *optical networks* should also be considered reflecting the fact that the proposed framework can support bufferless networking, thus it has the ability to eliminate the expensive power-hungry line cards and to build all-optical cross-connects. A more detailed discussion about the application and deployment options can be found in Section 7.2.

3.3 DFCP: Fountain Coding in the Transport Layer

This section is devoted to introduce the *Digital Fountain based Communication Protocol (DFCP)*, and to describe the main design principles and implementation issues. First, a brief overview of DFCP is given, then its operating mechanism is discussed including the protocol header structure, the connection establishment and termination processes, the coding and data transfer method, as well as flow control. Since DFCP is under continuous research and development, we close the section with adjustable protocol-specific parameters intended to facilitate future experiments.

3.3.1 Overview

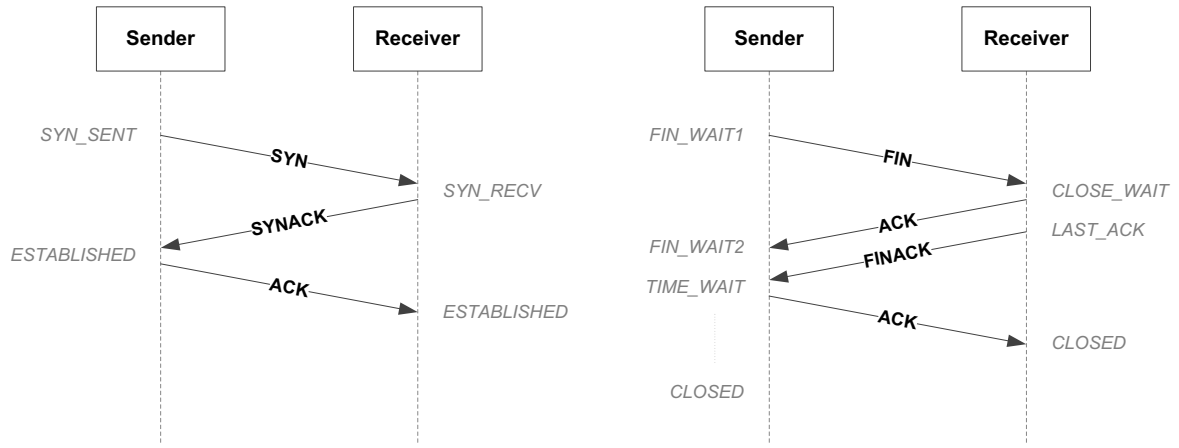
DFCP is a connection-oriented transport protocol, which can be found in the transport layer of the TCP/IP stack, and it ensures *reliable end-to-end communication* between hosts like TCP. The operation of the protocol consists of four main steps, namely connection establishment, coding, data transfer and connection termination. However, unlike TCP our protocol does not use any congestion control algorithm, but just encodes the data using Raptor codes and sends the encoded data towards the receiver at the maximum possible rate yielding a very efficient operation. In this case, efficient means that available resources in the network can be fully and quickly utilized without experiencing performance degradation. Although coding and decoding need extra overhead, it will be shown in Chapter 4 that this approach has many advantages and can eliminate several drawbacks of TCP.

DFCP has been implemented in the Linux kernel version 2.6.26-2. Similar to TCP, the interaction between the applications and our transport mechanism is handled through the socket layer using the standard system calls. The socket structure associated with DFCP stores all protocol-specific information including flow control and coding settings.

3.3.2 Protocol Header

The protocol header can be seen in Figure 3.5 including the name of each field and its size in bits. The source and destination ports give the port numbers used for the communication between the sender and receiver applications. Since packets are organized into blocks, the block ID identifies the block which the given packet belongs to. The

Figure 3.5. Protocol header structure



(a) Creating a new connection (b) Closing an existing connection

Figure 3.6. The connection establishment and termination processes

fields S1, S2 and S3 contain 32-bit unsigned integers, which play roles in the encoding and decoding processes. The offset gives the number of 32-bit words in the header, and hence specifies where the first bit of the application data can be found. Flags (e.g. *SYN*, *FIN*) are primarily used in the connection establishment and termination phases, which are discussed in detail in the following subsection. The checksum is a generated number depending on the content of the header and partially on the data field.

3.3.3 Connection Establishment and Signaling

DFCP's connection establishment is based on a *three-way handshake* procedure (see Figure 3.6) as in the case of TCP [1]. The handshaking mechanism is designed so that the sender can negotiate all the parameters necessary for decoding with the receiver before transmitting application data. When the data is successfully received by the destination host, the connection is released similarly to TCP.

Creating a Connection

- **Step 1.** First, a *SYN* segment is sent to the destination host including the information used in the decoding process at the receiver side, and a timer is started with a timeout of 1 second. After transmitting the *SYN* segment, the sender gets into *SYN_SENT* state. If no reply is received before the timeout expires, the *SYN* segment is retransmitted and the timeout is doubled. After 5 unsuccessful retries, connection establishment is aborted, and the resources are released at the sender.

- **Step 2.** If the *SYN* segment is received by the destination host, it gets into *SYN_ RECV* state and sends back a *SYNACK* segment to the source host. The *SYNACK* message also contains information for the coding process, and it is retransmitted a maximum of 5 times if necessary as in the case of the *SYN* segment.

- **Step 3.** After receiving the *SYNACK* segment, the source host sends an *ACK* segment to the destination and gets into *ESTABLISHED* state. When the *ACK* is received by the destination, it also gets into *ESTABLISHED* state indicating that the connection is successfully made. If the *ACK* segment is lost, it can be detected at the sender by receiving *SYNACK* again. When the *SYNACK* message cannot be delivered 5 times, the connection is closed by an *RST* segment.

Closing the Connection

- **Step 1.** When one of the hosts wants to terminate the connection, it sends a *FIN* segment to the other side, and the state of the sender is changed to *FIN_ WAIT1*. Similar to the connection establishment phase, a timeout and retransmitting are used for *FIN* messages. If the acknowledgment is not received after 5 times of retry, the connection is closed and the resources are released.

- **Step 2.** The receiver sends an *ACK* message as a reply to the *FIN* segment and gets into *CLOSE_ WAIT* mode while the state of the sender is changed to *FIN_ WAIT2*. If the receiver also wants to close the connection, it sends a *FINACK* segment to the sender and gets into *LAST_ ACK* state. *FINACK* can be retransmitted a maximum of 5 times similar to the *FIN* message.

- **Step 3.** By receiving the *FINACK* segment, the sender gets into *TIME_ WAIT* state and sends an *ACK* message to the receiver. Since the receiver can retransmit the *FINACK* segment, it can be detected if the *ACK* segment is lost. After waiting in *TIME_ WAIT* state for a given time, the resources are released. When the receiver gets the *ACK* message, its state is changed to *CLOSED* and the resources are released at this side as well.

Since DFCP keeps the network congested due to the operation in the overloaded regime, important signaling messages and acknowledgments can be lost during the transmission. A possible way to handle this problem is giving high priority to these packets.

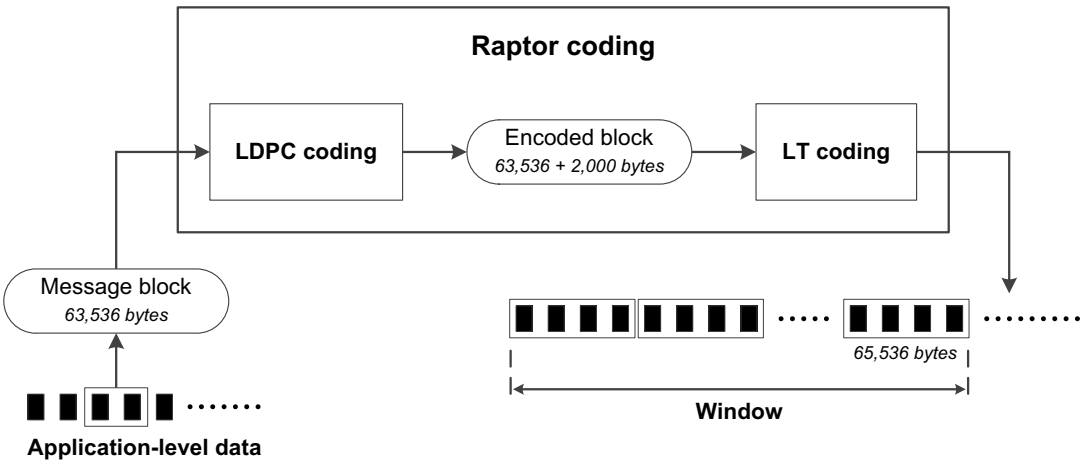


Figure 3.7. The flow chart of the coding and data transfer process

3.3.4 Coding Scheme

The flow chart of the coding and data transfer process can be seen in Figure 3.7. Once the connection is successfully established, the protocol is ready to send application-level data. First, the original data bytes received from the application are organized into message blocks and each of them is temporarily stored as a structure in the kernel memory before encoding. DFCP performs encoding for the stored message blocks sequentially, and once a given encoded block has been transferred to the receiver, the allocated memory is freed.

As shown in Figure 3.8, Raptor coding [46] involves two phases: *precoding* and *LT coding* [45]. In our implementation, precoding is realized by LDPC (Low-Density Parity-Check) coding [65], which adds some redundant bytes to the original message. The LT coder uses the result of the LDPC coding phase as input and produces a potentially infinite stream of encoded bytes.

The concept of *LDPC coding* is the following. Let us consider a bipartite graph having n_m nodes on the left side and n_c nodes on the right side. The nodes on the left and right sides are referred to as *message nodes* and *check nodes*, respectively. An example is

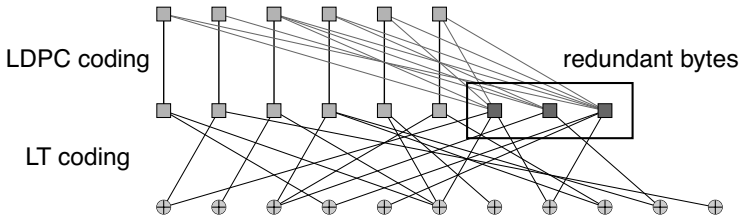


Figure 3.8. The encoding phases of message blocks

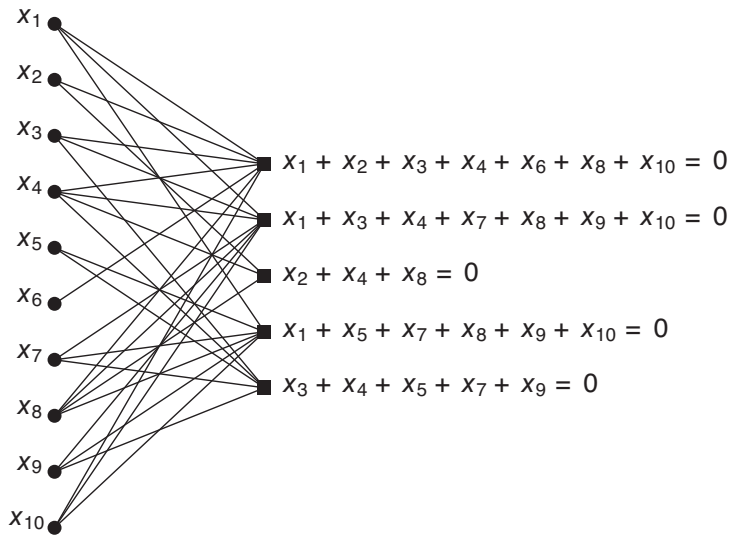


Figure 3.9. Example of an LDPC code

shown in Figure 3.9. As can be seen, for each check node it holds that the sum (XOR) of the adjacent message nodes is zero. In the latest version of the protocol, LDPC codes are generated by using a given probability distribution, and the initial value of the check nodes is set to zero. A specific degree d is calculated for each message node, which determines the number of its neighbors. After that, d check nodes are selected according to a uniform distribution. These check nodes will be the neighbors of the actual message node, and the new values of check nodes are computed as follows:

$$c_r = c_r \oplus m_i \tag{3.3}$$

where c_r denotes the randomly chosen check node and m_i is the actual message node. For example, as illustrated in Figure 3.9, degree $d = 2$ is chosen for the second message node x_2, and it is XORed with its neighbors, the first and the third check nodes. The value of a message node is associated with a byte of the original message. The LDPC encoder receives the application-level data in k bytes long blocks, which are extended by $n_c = n-k$ redundant bytes, and as a result the length of the encoded message will be n. In our implementation, the size of the original message block is $k = 63536$ and $n - k = 2000$ redundant bytes are added, thus the encoded length is $n = 65536$. If the application-level data is less than k, it will be padded with dummy bytes. It is an important part of the LDPC coding process that a random generator is used at both sender and receiver sides. The initial state of the random generator is determined by three variables (S1, S2 and S3), which are exchanged through the *SYN* and *SYNACK* segments.

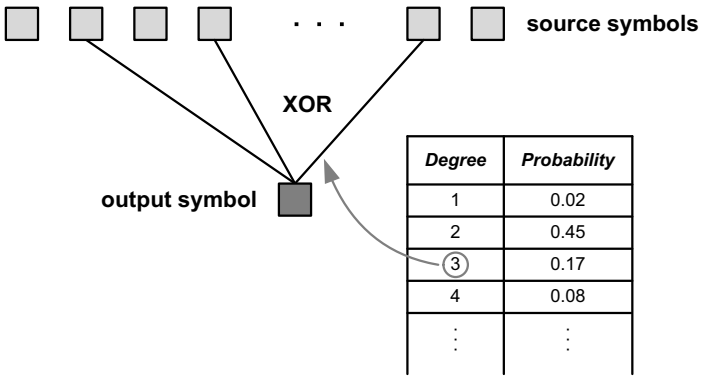


Figure 3.10. The concept of LT coding

The second phase of the Raptor coding scheme, *LT coding*, is performed on an encoded block of 65536 bytes received from the LDPC encoder. Figure 3.10 illustrates the LT coding process through a simple example. We have a given set of source symbols $x_1, x_2, \ldots, x_n$ (which correspond to single bytes in our implementation), and we would like to produce an encoded output symbol y. To this end, a degree distribution has to be given first, which defines how many source symbols will be used for generating the output symbol. After that, the following steps are performed:

- **Step 1.** A degree d is chosen based on the given degree distribution, which is equal to $d = 3$ in this example.
- **Step 2.** A specified number of random source symbols $r_1, r_2, \ldots, r_d$ are selected according to the previously chosen degree.
- **Step 3.** XOR operations are performed on the selected source symbols resulting in an encoded output symbol, that is $y = r_1 \oplus r_2 \oplus \cdots \oplus r_d = r_1 \oplus r_2 \oplus r_3$.

This procedure generates a single encoded byte that can be repeated as many times as needed. Finally, the LT encoder provides an encoded byte stream as output, which is then organized into 65536 bytes long encoded blocks. Since the actual state of the random generator depending on the initial state and the block ID is included in the protocol header, decoding at the receiver can be performed successfully.

3.3.5 Data Transfer and Flow Control

In order to prevent buffer overflows at the receiver side, we introduce a simple flow control mechanism by using a sliding window (see Figure 3.7). The sender is allowed to send a

certain number of LT encoded blocks, specified by the window size, without waiting for acknowledgments. Each encoded block is divided into packets and the encoded data is sent to the receiver packet by packet for all blocks found in the window. The size of a DFCP packet extended with the protocol headers is close to the MTU. During the transmission, the sending rate is controlled at the source host according to the result provided by the bandwidth estimation algorithm. The data transfer process continues until an acknowledgment has been received for the given block allowing the user application to send the next encoded blocks. This procedure guarantees that even if a large number of packets are lost, the receiver is able to restore the original message. As soon as the receiver has collected a sufficient number of LT encoded bytes (arriving in packets), it sends an acknowledgment for the received block to the sender. If the acknowledgment has been lost, the receiver resends it when additional packets are received from the same block. To ensure in-order delivery, DFCP assigns a continuously increasing unique identifier to each block in the protocol header, hence the receiver can recover the original order of blocks automatically.

We mention that, until a block ACK travels back to the sender, it produces and transmits additional encoded symbols which are not useful for the receiver, and this phenomenon is more pronounced in high BDP networks. However, we emphasize that any kind of reliable, feedback based transport mechanisms (including TCP) suffer from similar issues causing performance degradation or low network utilization. In comparison with TCP, DFCP utilizes available resources more efficiently at the price of this factor, but its impact can be mitigated in several ways. For example, acknowledgments can be sent immediately by the receiver when enough encoded symbols are received even if decoding has not been performed yet. In the case of RaptorQ [50], which is currently the most efficient variant of Raptor codes, only two additional symbols can provide a successful decoding probability greater than 99.9999%. Another possible way is to collect statistics about some relevant network parameters such as link delay and packet loss rate, and to calculate the expected number of encoded symbols to be sent, which will probably be sufficient for decoding at the receiver. The main advantage of this approach is that the sender can stop the transmission of encoded symbols without waiting for an ACK, and additional symbols are required only in the case when the link characteristics change abruptly (e.g. the loss rate gets significantly higher than the estimated value). Decoding failure is very rare [50], but when it occurs the extra packets received in the meantime will be enough for a successful outcome. Moreover, the block size can also be flexibly set

in a wide range, which could lead to more efficient operation in some applications (e.g. long data transfers) as the number of unnecessarily sent symbols can be reduced.

3.3.6 Main Parameters

The recent version of DFCP offers a number of ways for experimentation through the following adjustable protocol-specific parameters:

- **Window size.** It controls the maximum number of LT encoded blocks within the sliding window. The receiver acknowledges each block, but the sender is allowed to send all blocks of a window without waiting for acknowledgments.
- **Redundancy.** It gives the total redundancy (in percentage) added to the original message by both the LDPC and LT coders. The lowest possible value of this parameter depends on the applied coding scheme. In general, the lower the value, the more useful data can be transmitted from source to destination assuming a given link capacity.

The main goal of our research is to investigate the performance aspects of the digital fountain based data transfer paradigm. The use of Raptor codes is only one possible option for encoding data, hence the proposed concept is not restricted to the type of fountain code and is open for its future evolution. To enable the separation of the coding process from the transport mechanism itself, the different coding phases (encoding/decoding) and ACKs can be switched ON or OFF independently of each other for testing purposes.

3.4 Evaluation Methodology

In practice, performance evaluation of a transport protocol requires using different tools to get a clear picture about its behavior and specific properties, and to draw right conclusions. Even so, most researchers choose only one way to investigate their proposed protocols, namely simulation or testbed measurements. Especially for novel protocols and algorithms it can be misleading due to the unique nature of such environments. On the one hand, the main risk of relying only on simulation results is the fact that simulation environments are far from realistic in most cases, thus many real-world factors can easily be neglected [66, 67]. On the other hand, performing only testbed measurements can also lead to the loss of generality, because special hardware components can affect the results. In addition,

building a network testbed is a time-consuming process, and measurements are often very difficult to repeat [68, 69].

Since DFCP is based on a novel paradigm, it is crucial to ensure that our performance analysis results are reliable and the conclusions are valid. In order to fit these requirements, the measurements were carried out on *multiple platforms* including our laboratory testbed, the Emulab network emulation environment [8] and the ns-2 network simulator [7]. First, this section describes the performance metrics used for the evaluation, and after that the network topologies and scenarios are presented. Finally, a description of the different platforms is given focusing on the configurations, settings and parameters used in the analysis.

3.4.1 Performance Metrics

To evaluate the performance of transport protocols, there are some well-known metrics in the literature. One of the most widely used measures is *throughput*, which gives the amount of data successfully transferred per second from source to destination [70]. However, in many cases —especially if we compare the efficiency of transport mechanisms based on different principles— it is better to investigate *goodput* instead of throughput, because it refers only to the useful data bytes excluding the protocol headers, the added redundancy and the coding overhead. Therefore, in our measurements goodput was used as a primary performance metric. In the case of our proposed network architecture built upon DFCP, the analytical calculation of the goodput is feasible for simple scenarios. For example, consider a dumbbell topology (Figure 3.11) with a single bottleneck link of capacity c_B and N senders having access link capacities $c_1, c_2, \ldots, c_N$. Each sender transfers one flow simultaneously that results in N concurrent flows competing for the shared bottleneck capacity. Assuming that fair schedulers are used in the network routers, and the redundancy is denoted by ε, the goodput of flow i can be given as follows:

$$G_i = \begin{cases} \frac{c_B}{(1+\varepsilon_i)N} & \forall j : c_j \geq \frac{c_B}{N} \\ \frac{c_i}{1+\varepsilon_i} & c_i < \frac{c_B}{N} \\ \frac{c_B - \sum_{k=1}^{N} I_{\{c_k < \frac{c_B}{N}\}} c_k}{(1+\varepsilon_i) \sum_{k=1}^{N} I_{\{c_k \geq \frac{c_B}{N}\}}} & \exists j : c_j < \frac{c_B}{N} \text{ and } c_i \geq \frac{c_B}{N} \end{cases} . \tag{3.4}$$

Beyond measures related to the transfer rate, *flow completion time* (FCT) also serves as an important metric since most of the applications use flow transfers and the users' main interest is to download their flows as fast as possible [37]. FCT is the time elapsed from when the first packet of a flow is sent until the last packet is received. Flows transmitted via the Internet have very complex characteristics [36], and the mechanisms of different transport protocols can handle them differently. For example, it is known that TCP enters the congestion avoidance phase after slow-start, which takes many round-trip times, but the majority of short-lived flows never leave slow-start resulting in a high FCT. In the case of long-lived flows the additive increase of the congestion avoidance phase limits the transfer speed, and the fact that TCP fills the bottleneck buffer also contributes to the increase of FCT and it is far from being optimal.

Fairness is also an important property of transport protocols describing how they behave in a situation when two or more flows compete for the available bandwidth of a bottleneck link [71]. In our experiments we used the Jain's index as the fairness measure, which is a widely accepted fairness index in the literature [24]. Jain's index can be calculated by the following formula:

$$JI = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{3.5}$$

where x_i denotes the throughput (or goodput) of flow i and n is the number of concurrent flows. It returns a value between 0 and 1 where a higher value indicates a higher degree of fairness.

3.4.2 Network Topologies and Scenarios

The performance of DFCP was evaluated on different network topologies including the simple dumbbell topology and the more complex parking lot topology frequently used in the literature for experiments [72]. The dumbbell topology consisting of N source-destination pairs can be seen in Figure 3.11 where the data is transmitted from S_i to D_i in flow i. First, we experimented with a single flow ($N = 1$) to reveal the ability of DFCP to resist against varying *delay* and *packet loss rate* values of the connection. In this case, the bottleneck link capacity (c_B) was set to 1 Gbps. Furthermore, we studied the *fairness properties* of DFCP by using two source and destination nodes ($N = 2$). The main purpose was to observe how DFCP behaves in a situation when two concurrent flows compete for the available bandwidth determined by the bottleneck link. In this scenario, both the

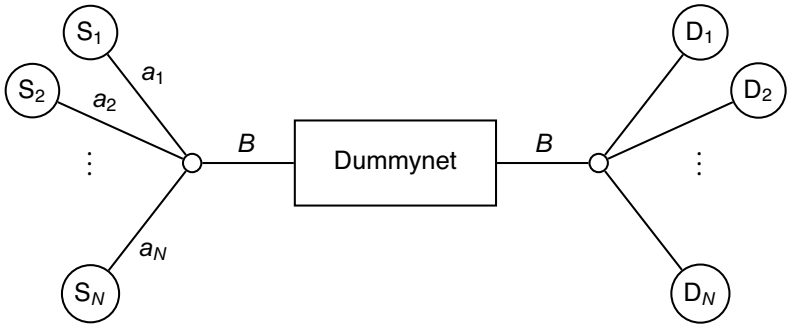


Figure 3.11. Dumbbell topology with N source-destination pairs

access links (a_1, a_2) and the bottleneck link (B) had a capacity of 1 Gbps. Regarding *scalability*, we investigated the performance and fairness stability of DFCP for increasing number of flows ($N = 10, 20, \ldots, 100$) and bottleneck bandwidth ($c_B = 0.1, 1, 10$ Gbps).

The scenarios described above made it possible to explore the fundamental features of DFCP and its scalability. Beyond these experiments, DFCP was studied in a more realistic environment as well. Figure 3.12 depicts a parking lot topology with three sender and receiver nodes, which contains two bottleneck links. In a real network multiple bottlenecks are common, and therefore, it is indispensable to evaluate how a transport protocol performs in such conditions. In these tests, the capacity was 1 Gbps for each access link (a_1, a_2, a_3), and the bottleneck link capacities (c_{B_1}, c_{B_2}) were set to different values as discussed in the following chapters.

Measurements lasted for 60 seconds in most scenarios (except if mentioned otherwise), and the results were obtained by excluding the first 15 seconds in order to ignore the impact of transient behavior of the investigated transport protocols. Regarding the scheduling discipline, WFQ (Weighted Fair Queuing) was applied by default in the intermediate nodes with equal weights [73]. However, we also experimented with other fair schedulers like DRR (previously suggested for our paradigm) [56] and SFQ (Stochastic

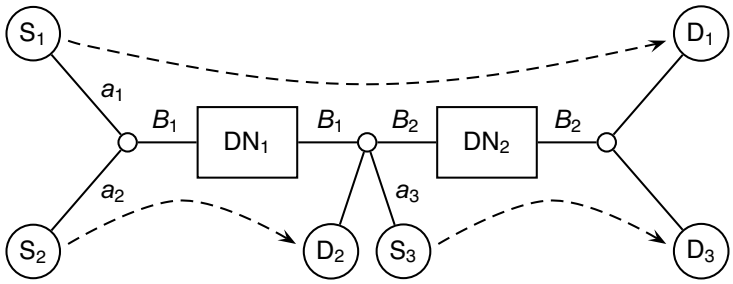


Figure 3.12. Parking lot topology with three source-destination pairs

Table 3.1. Hardware components of our laboratory test computers

(a) Hardware components of senders and receivers

Component	Type and parameters
Processor	Intel® Core™2 Duo E8400 @ 3 GHz
Memory	2 GB DDR2 RAM
Network adapter	TP-Link TG-3468 Gigabit PCI-E
Operating system	Debian Lenny with modified kernel

(b) Hardware components of network emulators

Component	Type and parameters
Processor	Intel® Core™ i3-530 @ 2.93 GHz
Memory	2 GB DDR2 RAM
Network adapter	TP-Link TG-3468 Gigabit PCI-E
Operating system	FreeBSD 8.2

Fair Queuing) [74], as well as with FIFO scheduler (using the DropTail queue management policy) [75] which is the simplest algorithm available in today's network routers.

3.4.3 Test Environments

Performance evaluation was conducted on the following three different platforms independently, and here we give a brief description of each:

- **Laboratory test network.** The *laboratory testbed* consisted of senders, receivers and a Dummynet network emulator [76], which was used for simulating various network parameters such as queue length, bandwidth, delay and packet loss probability. Each test computer was equipped with the same hardware components according to Table 3.1.

- **Remote emulation environment.** Our second testing platform was *Emulab*, which is a network testbed giving researchers a wide range of environments in which to develop, debug and evaluate their systems [8]. The Emulab architecture consists of two control servers (called boss and ops), a pool of physical resources that are used as experimental nodes (generic computers, routers or other devices) and a set of switches that interconnect the nodes. The boss server provides a graphical interface to the users and controls the internal operation of the test network. The experimental nodes can be accessed through a login shell running on the ops server.

Table 3.2. Hardware components of the Emulab test computers

(a) Hardware components of senders and receivers

Component	Type and parameters
Processor	Intel® Xeon® processors @ 3 GHz
Memory	2 GB DDR2 RAM
Network adapter	Intel Gigabit PCI-E
Operating system	Debian Lenny with modified kernel

(b) Hardware components of network emulators

Component	Type and parameters
Processor	Intel® Xeon® E5530 @ 2.40 GHz
Memory	12 GB DDR2 RAM
Network adapter	Broadcom NetXtreme II 5709 Gigabit PCI-E
Operating system	FreeBSD 8.3

In order to design and perform different experiments, the virtual network topology has to be defined by a TCL (Tool Command Language) description. Our measurement setup in Emulab was identical to the one used in our laboratory testbed for each test scenario, but the test machines were equipped with different hardware components as summarized in Table 3.2. The type of the sender and receiver nodes was *pc3000* according to the Emulab label system, and the network emulators were run on *d710* type nodes. As done in the case of laboratory measurements, our modified kernel including the implementation of DFCP was loaded into the sender and receiver machines.

- **Simulation framework.** Beyond the real testbeds described above, we also used the widely known *ns-2 network simulator*, which provides a powerful tool for researchers to try out and evaluate their new methods [7]. Since the prototype of DFCP has been implemented in the Linux kernel, we had to find a way to make the simulation of our protocol possible directly through the network stack of Linux. In fact, there are some tools available for this purpose, but only few of them can provide reasonable accuracy and efficiency, as well as support a wide range of operating systems and kernel versions [77]. Focusing on these requirements, after careful consideration *Network Simulation Cradle (NSC)* was chosen, which is an extension for wrapping kernel code into simulators allowing the investigation of real-world behavior [6]. NSC supports the simulation of network stacks of many operating

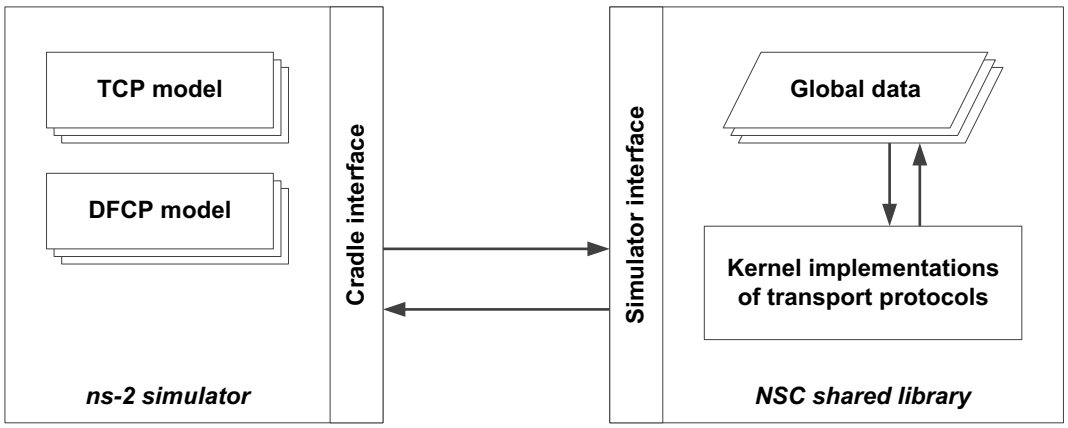


Figure 3.13. The DFCP-compatible integrated simulation framework

systems such as FreeBSD, OpenBSD, lwIP and Linux. Regarding reliability, it has been validated by comparing situations using a test network with the same situations in the simulator, which showed that NSC is able to produce extremely accurate results. However, NSC only enables the simulation of TCP versions and new TCP-like transport mechanisms, hence several protocol-specific modifications have been made to integrate the source code of DFCP into the framework. Figure 3.13 shows the main elements of the integrated simulation environment. The basic models of transport protocols are defined in ns-2 including all necessary parameters. The two simulator components (ns-2 and NSC) communicate through a common interface, provided by a C++ shared library. In case of an interaction, ns-2 invokes the related protocol-specific methods in NSC, which then call the proper kernel function. NSC can handle multiple copies of the global data used by the network stack making possible to run independent instances of protocol implementations within the same simulation scenario.

3.5 Fundamental Properties

In this section, we first study and validate the fundamental properties of DFCP on the three different testing platforms discussed above. We also quantify the impact of the main parameters of DFCP on the goodput performance, as well as present and discuss the consequences of the dead packet phenomenon by investigating an SDN-based rate control mechanism.

Table 3.3. Goodput performance in Mbps for different network parameters

Platform	**Packet loss rate**				**Round-trip time**		
	0.1%	*1%*	*5%*	*10%*	*0 ms*	*10 ms*	*50 ms*
Testbed	730	690	623	562	791	791	774
Emulab	773	718	649	583	821	821	821
ns-2	755	720	677	631	842	842	842

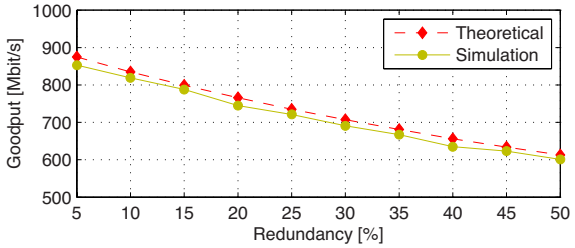
3.5.1 Operation under Different Network Conditions

Table 3.3 presents the main features of DFCP introducing its high resistance to varying network conditions such as *packet loss rate* and *round-trip time*. In our experiments, unless mentioned otherwise, we used a uniform loss model with random, independent packet losses. These measurements were carried out on a dumbbell topology with one source-destination pair (see Figure 3.11). It is known that TCP is very sensitive to packet loss resulting in a quick performance degradation for increasing loss rate. The table clearly indicates that DFCP can operate efficiently even in high loss rate environments with only a negligible decrease in goodput. The table also illustrates that the goodput performance of DFCP is independent of the round-trip time.

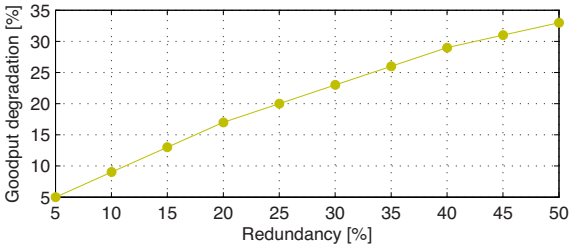
Considering *intra-protocol fairness*, DFCP can ensure equal bandwidth sharing among concurrent traffic flows thanks to the use of fair schedulers in the routing nodes. Our measurements conducted on dumbbell and parking lot topologies also confirmed this beneficial property.

3.5.2 Effect of Protocol-Specific Parameters

Redundancy, denoted by ϵ, highly determines the efficiency of fountain coding schemes since a lower value makes it possible to transmit more useful data bytes at a given link. Figure 3.14 demonstrates how the redundancy parameter of DFCP affects the goodput performance when the window size is set to 1000 blocks. The theoretical curve of Figure 3.14a is derived from the goodput formula (3.4) defined in Section 3.4 by taking into account the overhead (i.e. protocol headers) at different layers as well. One can see that ns-2 simulation results fit well to the theoretical values. Figure 3.14b shows the goodput degradation of DFCP as the amount of redundancy increases. If the redundancy is about 5%, it leads to approximately the same degree of performance degradation. However, the decrease in goodput does not change linearly with increasing redundancy. For example, 50% of redundancy wastes only 33% of the maximum bandwidth, which can be utilized



(a) Goodput performance for increasing redundancy



(b) Performance degradation for increasing redundancy

Figure 3.14. The impact of the redundancy parameter

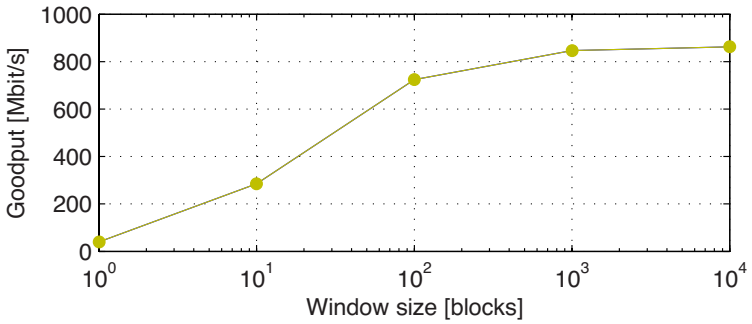


Figure 3.15. The impact of window size on the goodput performance

for useful data transmission. In practice, the typical value of redundancy is below 5% for recent fountain codes [46].

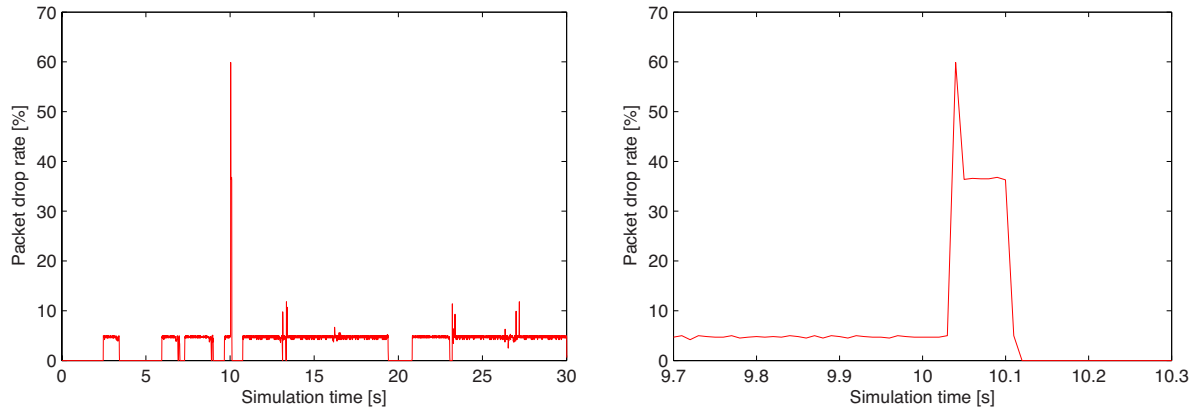
Figure 3.15 illustrates the impact of DFCP's flow control with $\epsilon = 0.05$, which can be controlled by the *window size* parameter. As mentioned in Section 3.3, the window size is measured in LT encoded blocks. The figure shows that, as the window size increases, a higher goodput can be realized. Since the Raptor coding scheme can generate an infinite stream of encoded bytes, in theory it is plausible to choose a window size as high as possible. However, there are two aspects should be taken into consideration. First, flow control is used to prevent buffer overflow at the receiver end. Secondly, the use of a larger window leads to a more bursty traffic. In general, it is practical to limit the window size at the point where further increasing does not improve goodput, but delay-sensitive applications may require smaller windows.

3.5.3 Analysis of the Dead Packet Phenomenon

In Section 3.2.2, we pointed out that our mechanism requires the proper control of transmission rate in order to cope with the so-called *dead packets* [59], and hence, to minimize

the extent of bandwidth waste. This subsection focuses on how to leverage the capabilities of SDN to solve this issue and also quantifies the impact of the dead packet phenomenon. In the last years, as the SDN paradigm [78] becomes more and more decisive in the networking industry, a significant research effort has been devoted to explore the benefits it can bring in comparison to traditional computer networks. One of the areas where the SDN architecture opens new horizons is *network monitoring*. Although passive and active measurement techniques have a long research history (see Section 3.2.2 for a brief overview), the central knowledge of SDN controllers can help to design much more efficient and accurate monitoring tools, therefore it is a very active research topic being in the focus of many papers and ongoing works. For example, FlowSense [79] measures link utilization in a non-intrusive way by analyzing the control messages between the switches and the controller. Due to the fact that SDN controllers know both the topology and the link capacities, the available bandwidth can easily be computed. Another framework called PayLess [80] can deliver highly accurate information about the network in real-time without incurring significant overhead whereas OpenNetMon [81] exploits OpenFlow [82] to provide per-flow metrics including throughput, delay and packet loss. Authors of [83] present a software-defined transport (SDT) architecture for data center networks in which a central controller computes and sends flow rates periodically to hosts enabling real-time rate control in a scalable way.

To quantify the bandwidth wasted due to the greedy transmission mechanism of DFCP, we carried out some experiments assuming that an SDN-based solution is used to estimate



(a) Drop rate over a 30 seconds long measurement interval

(b) Drop rate in case of sudden bandwidth change (enlarged view)

Figure 3.16. Packet drop rate at the bottleneck router using SDN-driven rate control (simulation)

Table 3.4. Packet drop rate for different response times and estimation error (simulation)

Response time	**Drop rate at the bottleneck router**		
	1% error	*5% error*	*10% error*
5 ms	0.58%	3.32%	7.74%
10 ms	0.59%	3.37%	7.81%
50 ms	0.63%	3.48%	7.97%
100 ms	0.69%	3.65%	8.19%

the available bandwidth and to control the rate at the sender. In software-defined networks the monitoring accuracy is mainly determined by the *polling frequency* and the *link delay* between the switches and the controller, which we call *response time* in the following. In the context of our concept, response time is interpreted as the time elapsed from a bandwidth change until rate adaptation is performed at the sender, which includes the polling and processing overhead, as well as the switch-to-controller and controller-to-sender communication delay.

Here we investigate a scenario on the parking lot topology illustrated in Figure 3.12 where the bottleneck links, B_1 and B_2, have a capacity of 1 Gbps and 400 Mbps, respectively. The link delays were set such that flows experienced a round-trip time of 50 ms on B_1 and 30 ms on B_2. In DFCP, the window size was adjusted to 1000 and we used a redundancy value of 5%. Assume that *flow 1* and *flow 2* start data transfer at the same time while *flow 3* launches 10 seconds later. Each sender can control its transmission rate with a given accuracy according to the information provided by the SDN-based available bandwidth measurement method. Figure 3.16 shows the packet drop rate at the second bottleneck router in the function of time for 5% estimation error and 50 ms response time. Before *flow 3* enters *flow 1* and *flow 2* receive 400 Mbps and 600 Mbps of B_1, respectively, because the available bandwidth is 400 Mbps along the path that *flow 1* traverses. When *flow 3* joins at the time of 10 seconds, the available bandwidth on the path of *flow 1* decreases to 200 Mbps since the scheduler shares the capacity of B_2 between *flow 1* and *flow 3* equally. At this point, a high instantaneous drop rate can be observed because the bandwidth is wasted until the sender reacts to traffic changes. Table 3.4 summarizes the mean drop rate calculated over a 30 seconds long measurement period from 10 runs for realistic parameter settings. The results suggest that estimation accuracy is an important factor whereas response time only slightly affects the drop rate.

Overall, we believe that in the case of any transfer mechanism including TCP and DFCP, a trade-off has to be found among different performance determining factors. In

fact, DFCP uses a very efficient transfer method but it pays the price in the dead packet phenomenon. However, this issue can be handled as shown in the above case study, and SDN offers a promising solution, which will be one of our future research directions.

3.6 Conclusion

This chapter presented a novel communication paradigm exploiting efficient fountain codes for error correction. Our main concept, which is built upon a network architecture and a data transfer mechanism, was introduced together with the potential benefits. Since the key component of the solution is a transport protocol called DFCP, its design and operating principles were discussed in detail as well. Moreover, we carried out a performance analysis in a multi-platform environment to explore the fundamental properties of digital fountain based transport. Our results demonstrated the high loss resistance of DFCP and its low sensitivity to delay. We also elaborated on the impact of protocol parameters and gave some guidelines about how to set them optimally.

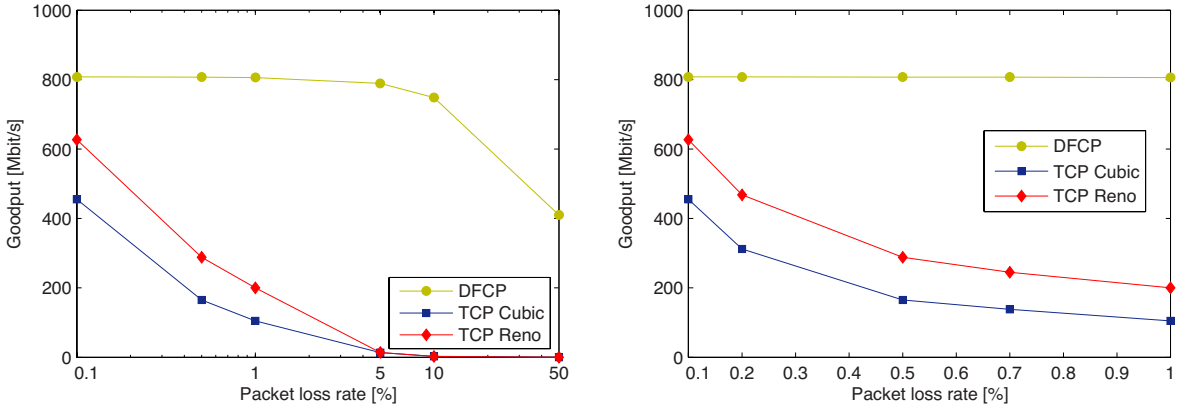
Chapter 4

Fountain Coding versus Congestion Control: A Comprehensive Performance Evaluation Study

In the previous chapter, we investigated a novel data transfer paradigm based on digital fountain codes as a possible alternative to congestion control. This chapter is intended to present a comprehensive performance evaluation of these two concepts under different network conditions. The experiments were carried out in a cross-validated test environment consisting of multiple platforms such as real testbeds and a packet-level network simulator. We focus on the steady-state behavior of transport mechanisms and reveal how it is affected by various network parameters. Finally, we close this chapter with our main conclusions regarding the advantages of the new proposal over the traditional congestion control based approach.

4.1 Steady-State Analysis

This section presents a comprehensive performance analysis study by comparing DFCP to different TCP versions, namely TCP Cubic which is the default congestion control algorithm in the Linux kernel and TCP NewReno with SACK option. Most of the measurements were performed on three testing platforms (testbed, Emulab and ns-2), and here we present representative results. Unless mentioned otherwise, the window size and redundancy parameters of DFCP were set to 10000 blocks and 10%, respectively. In single flow experiments, we used a bottleneck buffer of 1000 packets and a buffer of 10000 packets in other scenarios except network utilization and scalability measurements.



(a) A wide range of packet loss rates (b) A realistic parameter domain

Figure 4.1. The performance of DFCP and TCPs in a lossy environment (simulation)

4.1.1 Goodput Performance

In this part, we focus on the *goodput performance* of DFCP and TCPs they can provide in the long run. One of the main beneficial properties of DFCP can be seen in Figure 4.1. It demonstrates that DFCP is much more resistant to packet loss than TCP Cubic and TCP Reno. The difference in goodput is already considerable for 0.1% of packet loss, but for increasing loss rate DFCP highly outperforms both TCP variants (see Figure 4.1a). For example, for 1% of packet loss the ratio between the goodput obtained by DFCP and TCP Reno is about 4, and this ratio is almost 8 for TCP Cubic. When the loss rate attains 10%, DFCP gets more than 250 times faster compared to TCPs, and it works efficiently even in case of extremely high loss (50%) in contrast to TCPs, which are unable to operate under these network conditions. A practical result is shown in Figure 4.1b where the goodput is examined only in the interval 0.1–1%. The figure illustrates that, in realistic settings, DFCP becomes insensitive to packet loss, hence the rate variation experienced in the case of TCP can be avoided. Moreover, the goodput performance of DFCP is significantly better compared to both TCP versions in the whole range.

Figure 4.2 shows the performance comparison results of DFCP and TCPs for varying round-trip time. The figure illustrates that TCP versions perform better than DFCP in terms of goodput regarding the RTT interval 0–10 ms, but the difference is negligible and it is due to the coding overhead. Nevertheless, for delay values greater than 10 ms DFCP achieves significantly higher transfer rate compared to TCP Cubic and TCP Reno. Since the typical value of round-trip time in a real network exceeds 10 ms [84], DFCP can function more efficiently than TCP in such conditions.

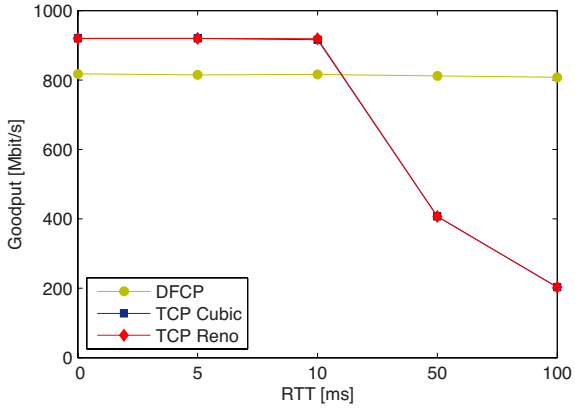


Figure 4.2. Goodput performance of a single flow for varying RTT (simulation)

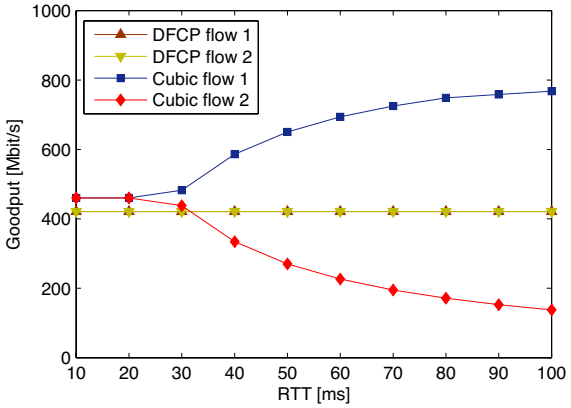
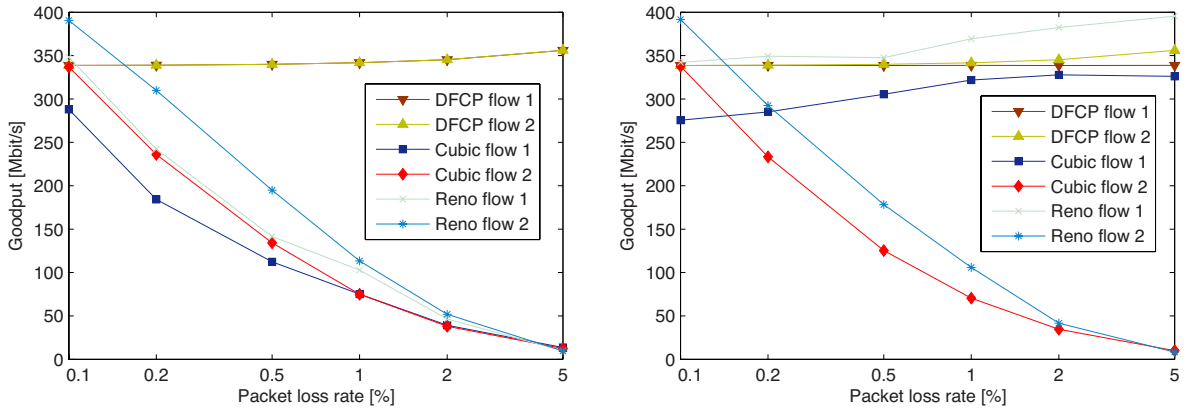


Figure 4.3. Bandwidth sharing of two competing flows (testbed)

Additionally, it is essential to reveal and investigate how a transport protocol shares the available bandwidth of a bottleneck link among competing flows often referred to as *fairness* property. As mentioned earlier, DFCP and TCP cannot work together within the same network due to the fact that they operate in different regimes according to the applied principles. For this reason, here we deal only with *intra-protocol fairness* analysis. As widely known, standard TCP cannot provide an equal portion of the bottleneck bandwidth for competing flows with different round-trip times [85] due to its AIMD mechanism [24]. Figure 4.3 depicts the goodput for two competing DFCP and TCP Cubic flows, respectively. The delay of *flow 1* was fixed at 10 ms, and for *flow 2* we varied the delay parameter between 10 and 100 ms. Since the results for TCP Reno were quite the same as in the case of TCP Cubic, only the latter was plotted. The figure shows that the bottleneck link capacity is equally shared by the two TCP flows for RTT values less than 20 ms in our testbed measurements. However, for RTTs greater than 20 ms the goodput of *flow 2* starts to decrease, and as a result, *flow 1* with lower RTT can gain access to a greater portion of the available bandwidth indicating the unfair behavior of TCP. In contrast, DFCP flows achieve perfect fairness as they share the bottleneck capacity equally and they are much less sensitive to the round-trip time compared to TCP. We note that the difference can be observed in the goodput of DFCP and TCP flows for RTT values less than 20 ms is due to the coding overhead.

Figure 4.4 illustrates the impact of packet loss rate on the goodput performance for two competing flows. Figure 4.4a shows the case when packet loss rates are equal for both flows and changed according to the horizontal axis, and Figure 4.4b shows the case when they experienced different loss rates. In the latter case, the first flow has a fixed loss rate



(a) Equal packet loss rate (b) Different packet loss rates

Figure 4.4. Goodput for two competing flows with equal and different packet loss rates (testbed)

set to 0.1%, and the second one has a loss rate varied between 0.1% and 5% as shown in Figure 4.4a. In Figure 4.4a, it can be observed that neither TCP Cubic nor TCP Reno flows share the available bandwidth equally for lower values of loss rate, however, the difference is reduced for increasing packet loss rate. Unlike different TCP variants, DFCP provides a fair resource allocation. On the one hand, each DFCP flow achieves nearly the same goodput value, and on the other hand it is almost independent of the packet loss rate. We note that the slight increase in goodput for higher loss rates can be attributed to some measurement artifacts. Figure 4.4b shows that, while DFCP behaves similarly in the cases of equal and different loss rates for the two flows, respectively, TCP Cubic and TCP Reno share the bottleneck link capacity in an unfair way in the whole range. We can conclude the robust property of DFCP, namely, it is irrelevant to DFCP that loss rates are equal or different for the competing flows, and what values they have.

Figure 4.5 presents the performance comparison of DFCP and TCP Cubic carried out on the parking lot topology illustrated in Figure 3.12 by starting three concurrent flows. In this test scenario, the capacity was set to 1 Gbps for both bottleneck links denoted by B_1 and B_2. The round-trip time was fixed at 10 ms on B_1, but it was increased on B_2 from 0 to 100 ms. Looking at the figure, we can make the following observations. Until the round-trip time experienced on B_2 attains 10 ms, both DFCP and TCP Cubic share the bottleneck bandwidth of B_1 and B_2 in a fair way. However, for higher delay values TCP Cubic gradually becomes unfair due to the fact pointed out in this section, namely, TCP is sensitive to round-trip time. As the goodput obtained by *flow 1* and *flow 3* drops for increasing RTT (since they go through B_2), *flow 2* with lower RTT receives more

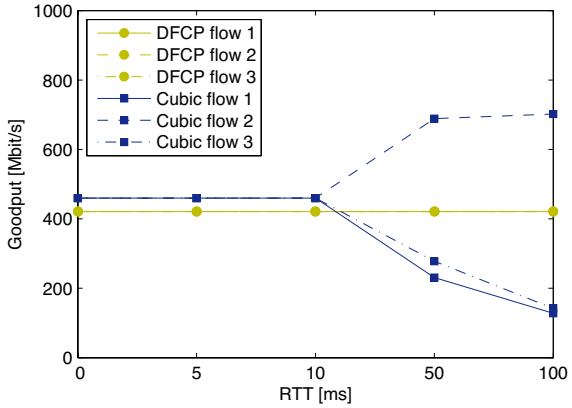


Figure 4.5. Bandwidth sharing in a multi-bottleneck network with varying delay (testbed)

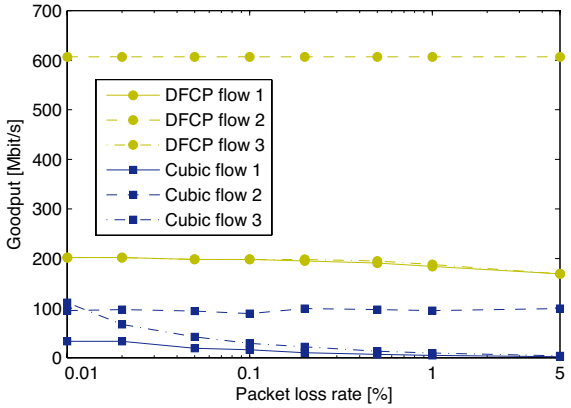


Figure 4.6. Goodput performance in a multi-bottleneck network with varying packet loss rate (testbed)

and more bandwidth. Accordingly, TCP Cubic does not provide fairness between *flow 1* and *flow 2* having different RTTs. Moreover, the available capacity of B_2 is also shared unequally, and hence, *flow 1* and *flow 3* achieve different goodput performance. As we mentioned earlier it is an undesirable behavior, and the results show that DFCP can resolve this issue by providing perfect fairness for each flow independently of their RTTs thanks to its robustness to varying network conditions.

Figure 4.6 demonstrates the results of a similar test scenario for varying packet loss rate performed on the same parking lot topology. In this case, the capacity was set to 1 Gbps and 500 Mbps for the bottleneck links denoted by B_1 and B_2, respectively. The packet loss rate was fixed at 0.01% on B_1, but it was increased on B_2 from 0.01% to 5%. The round-trip delay was set to 10 ms on both links. We can see that DFCP provides fair shares for the flows competing for the available bandwidth of B_2, and their goodput drops very slowly as the packet loss increases. Furthermore, the link utilization of DFCP is excellent on both bottleneck links due to its high loss resistance. In contrast, TCP Cubic and TCP Reno ensure fairness for *flow 1* and *flow 3* only for packet loss rate greater than 1% where both flows become almost unable to transfer data. The goodput of *flow 1* starts from a lower value than the goodput of *flow 3*, because *flow 1* goes through both B_1 and B_2, and hence experiences a higher rate of packet loss. The link utilization achieved by the investigated TCP variants is quite poor due to their sensitivity to this factor.

Overall, we can say that the goodput performance of DFCP is significantly better than in the case of the investigated TCP versions in a wide range of packet loss rates and round-trip times.

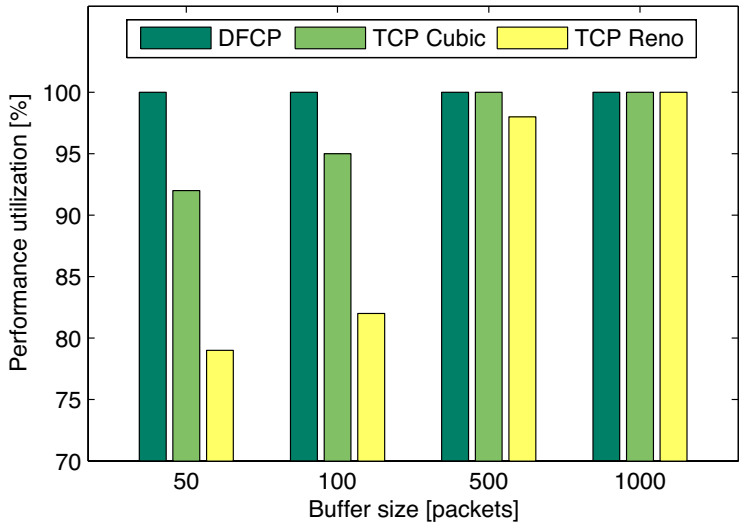


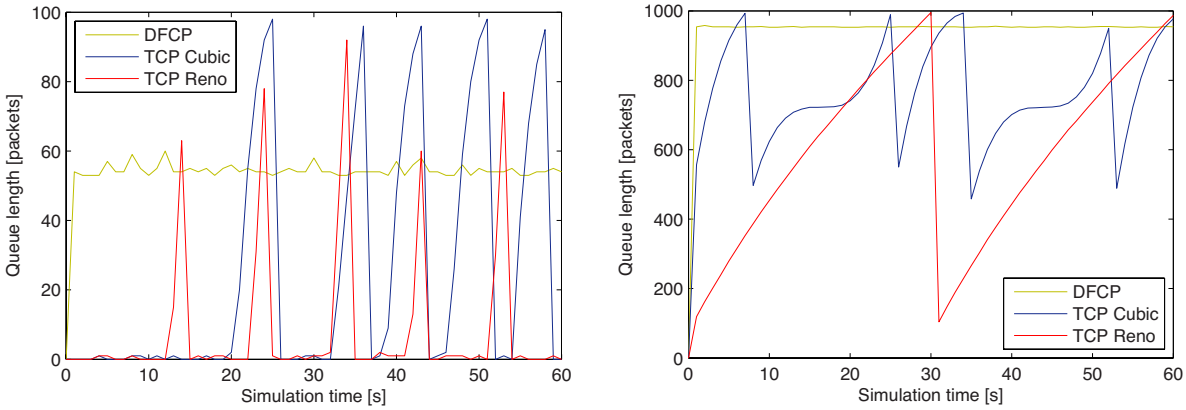
Figure 4.7. The impact of buffer size on the performance of DFCP and TCPs (simulation)

4.1.2 Buffer Demand and Occupancy

It is a well-known fact that the buffer size demand of TCP is at least of root order in the number of competing flows [86]. This requirement imposes a significant challenge in all-optical networks where only very small buffer sizes can be realized due to both economic and technological constraints [87].

Figure 4.7 demonstrates on the dumbbell topology how the performance of DFCP and TCPs is affected by the buffer size. In this scenario, the round-trip time was fixed at 10 ms and no packet loss was simulated. The buffer size is given in packets, and the vertical axis represents the performance utilization of the investigated transport protocols. *Performance utilization* is the ratio (expressed in percentage) between the goodput can be obtained with a particular buffer size and the maximum goodput that can be achieved when the buffer size is set as high as to exclude it from the limiting factors. We can see that, with a buffer size of 1000 packets, each protocol is able to realize maximum performance utilization. However, by decreasing the buffer size the performance of TCP variants drops considerably. For example, with a small buffer of 50 packets, TCP Cubic and TCP Reno can work only at a reduced transfer rate, 92% and 79% of the ideal case, respectively. In contrast, DFCP can bring out the maximum performance not only for large buffers, but also for small ones, and thanks to this property the transport mechanism of DFCP is closely aligned to the concept of all-optical networking [87].

Figure 4.8 illustrates how DFCP and different TCP versions utilize the bottleneck buffer. The average occupancy was calculated for a 600 seconds long interval. In the



(a) Buffer size: 100 packets (b) Buffer size: 1000 packets

Figure 4.8. Buffer occupancy (simulation)

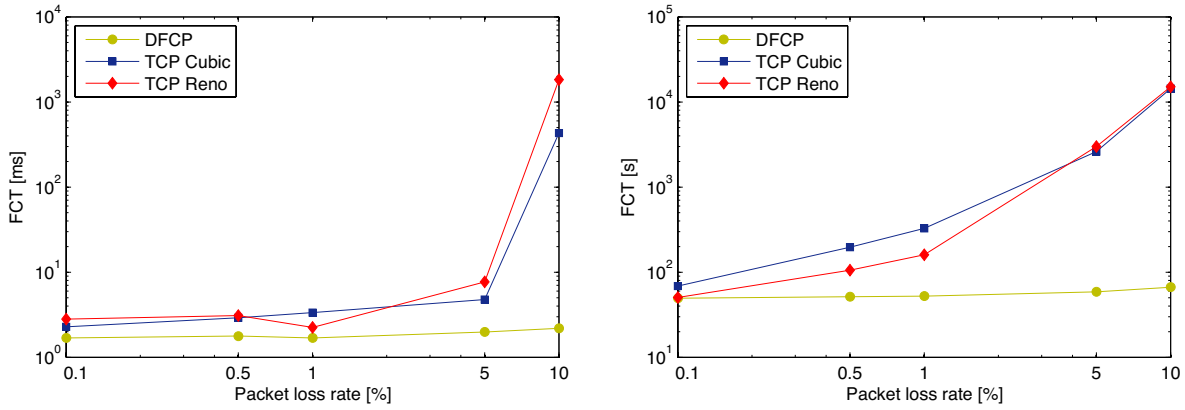
case of a small buffer of 100 packets (see Figure 4.8a) DFCP operates with an average utilization of 54% while TCP Cubic and Reno use only 43% and 11% of the available buffer space. Considering queue length dynamics, we can see that DFCP builds up the queue in a very short time, and then keeps it stable in contrast to TCPs. If the buffer can store 1000 packets (see Figure 4.8b), utilization becomes higher for each transport protocol. Specifically, DFCP works at a 95% buffer occupancy demonstrating that our concept is designed to fully saturate router buffers irrespective of their sizes. TCP Cubic and Reno also show considerable improvement in utilization as the average occupancy is 75% and 58%, respectively.

Our results revealed the robust property of DFCP regarding the buffer space demand, as it performs well both in small and large buffer environments without any oscillation phenomena usually observed in the case of different TCP versions.

4.1.3 Flow Transfer Efficiency

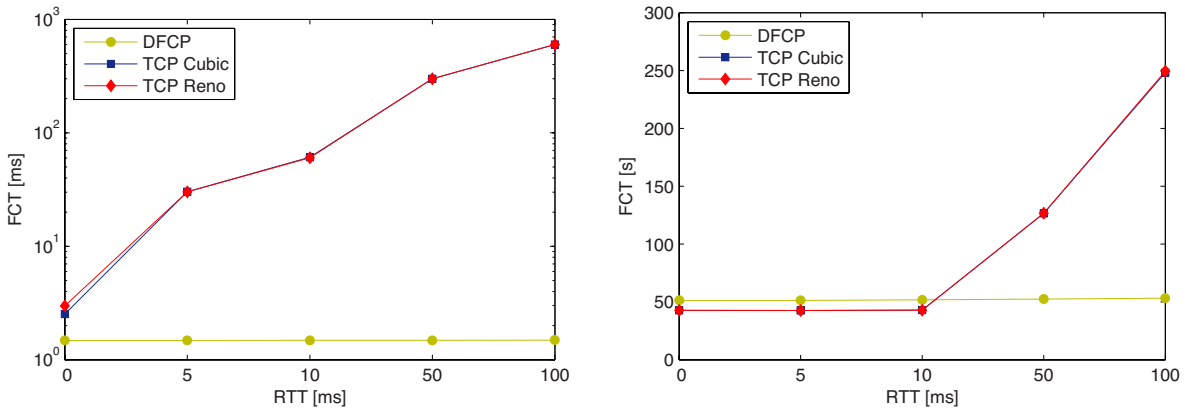
As we mentioned in Section 3.4, *flow completion time* is one of the most important performance metrics from the user's point of view because of the fact that users want to download web pages, softwares, movies and many other contents as fast as possible. Accordingly, we investigated two different categories: (1) web object (150 kB, the mean size is about 100–200 kB [88]) and (2) DVD (4.7 GB), which represent short and long data transfers, respectively.

Figure 4.9 illustrates how the flow completion time depends on the packet loss rate. One can see that DFCP provides the fastest download in both cases, indicating its potential



(a) Web object (b) Full DVD

Figure 4.9. Flow completion time for different packet loss rates (testbed)

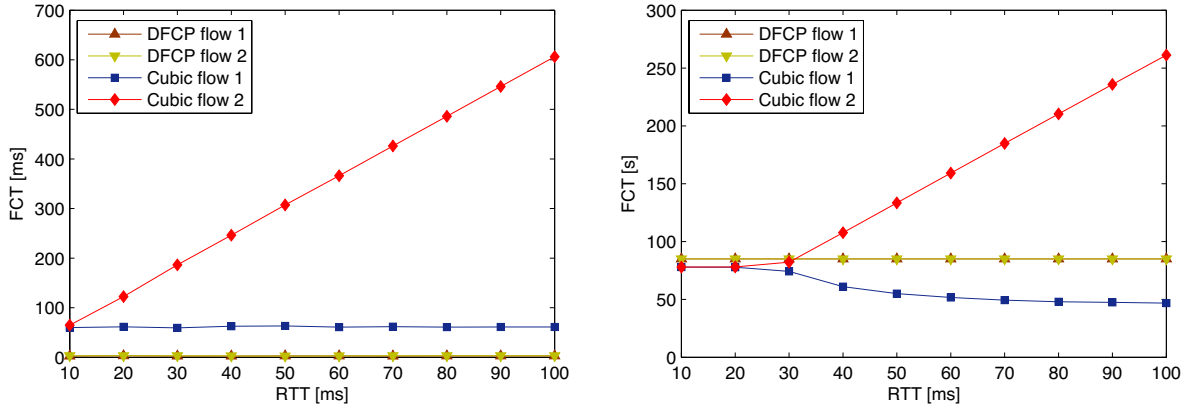


(a) Web object (b) Full DVD

Figure 4.10. Flow completion time for different round-trip times (testbed)

in the case of web traffic as well as heavy data transfers, however, the benefit is more significant in the latter case. By transferring a typical web object, the most considerable performance gain can be experienced for high packet loss rates (see Figure 4.9a). However, if we transfer a full DVD, the advantage of DFCP is pronounced in the whole range of packet loss rate (see Figure 4.9b). Moreover, DFCP becomes almost insensitive to packet loss in these practically relevant scenarios.

Investigating the impact of round-trip time, we can also find significant differences in the performance of DFCP and TCPs as shown in Figure 4.10. Specifically, in the case of a web object there are several orders of magnitude between the flow completion time of DFCP and TCPs for increasing round-trip delay (see Figure 4.10a). Considering the category of DVD, it can be stated that the difference in download time is negligible for



(a) Web object (b) Full DVD

Figure 4.11. Flow completion time for two competing flows with the one having a fixed RTT of 10 ms and the other one having an RTT varied between 10 and 100 ms (testbed)

low RTT values, however, it gets more and more significant towards high RTT values as shown in Figure 4.10b.

Figure 4.11 shows the flow completion time for two competing DFCP and TCP Cubic flows where the first flow has a fixed RTT of 10 ms and the delay of the second flow is varied between 10 and 100 ms. We observed that the results for TCP Reno were quite the same as in the case of TCP Cubic, hence only the latter was depicted. Looking at Figure 4.11a one can see that in the case of a web object DFCP produces excellent results. It does not only provide 20 times faster download than TCP even in the worst case, but also achieves perfect fairness, thus both DFCP flows have nearly the same download time. If we transfer a full DVD, the two TCP flows behave in a fair way, but only for RTT values less than 20 ms (see Figure 4.11b). In contrast, DFCP flows attain equal download time in the whole range since DFCP protocol is insensitive to high RTTs compared to TCP. We note that the difference in the flow completion times of DFCP and TCP flows for RTT values less than 20 ms is due to the coding overhead of DFCP.

The important issue of efficient flow transfer regarding different transport protocols is addressed in this subsection. The results demonstrate that the currently used TCP versions cannot achieve optimal performance in the case of short-lived and long-lived flows. In many applications such as web browsing it is a real limiting factor, and the user experience would be significantly improved by using a much more effective data transfer mechanism like DFCP.

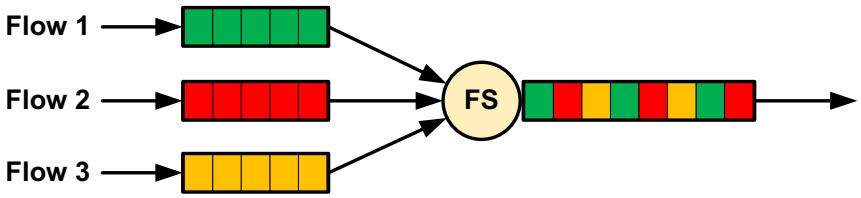
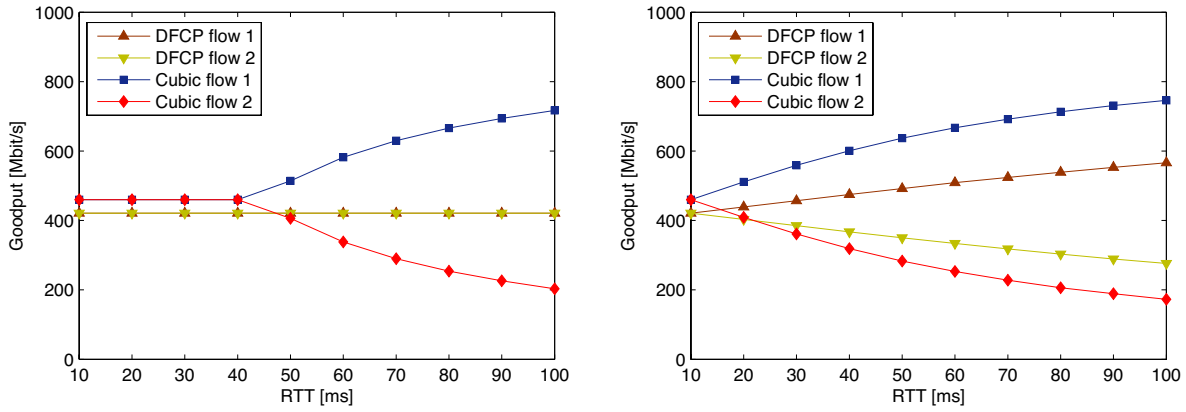


Figure 4.12. The general concept of per-flow fair scheduling

4.1.4 Fairness Properties

In our proposed future network architecture, *fairness* can be realized by the application of fair schedulers as we mentioned in Section 3.2. In fact, unlike TCP the transfer mechanism of DFCP cannot guarantee fairness at the host side. Therefore, the only way is to perform this task by network routers. However, in this context there are some open questions to be answered. On the one hand, a plenty of fair scheduling algorithms have been worked out during the last two decades, but only a few are available in today's routers and their impact on Internet performance is still poorly understood [89]. So, the natural question is which one to choose? On the other hand, in most routers a FIFO scheduler is applied by default as it is the simplest algorithm, but it does not eligible for providing fairness. How does DFCP perform in such conditions? To answer these questions, we extend our fairness analysis by investigating different queuing mechanisms. Since the fairness properties of SFQ and WFQ were very similar, we show results only for the latter.

About one decade ago, researchers pointed out that the pretty old DRR algorithm [56] can provide a much more efficient way for bandwidth sharing among concurrent flows than it thought before. According to a flow-level traffic model developed by Kortebi et al. [57], the number of flows that need to be handled by a per-flow scheduler is typically low and does not increase with link capacity. Measurements taken from commercial networks also confirmed this observation showing that the number of these active flows attains only several hundreds even in the case of high-speed connections [58]. Since we propose DRR as the primary queuing mechanism for our digital fountain based architecture, here we introduce its main principles and operating mechanism. The general concept of *per-flow fair scheduling* is illustrated in Figure 4.12 where three flows share a common bottleneck link. The packets of each flow are buffered in separate FIFO queues and serviced according to the given fair scheduling (FS) algorithm. DRR works in a round-robin fashion meaning that queues are serviced one after another in each round. A variable called deficit counter d_i is maintained for each per-flow queue, which determines the number of bytes that can be transmitted from queue i in the next round.



(a) DRR (b) FIFO

Figure 4.13. Bandwidth sharing with different queuing mechanisms (simulation)

The algorithm performs the following steps:

- **Step 1.** The initial value of deficit counters is set to zero, that is, $d_i = 0$ for each queue $i = 1, 2, \ldots, N$ where N denotes the number of queues.
- **Step 2.** A so-called quantum q is added to the deficit counter of per-flow queue i: $d_i := d_i + q$.
- **Step 3.** Let l be the length of packet p at the head of queue i. If $d_i \geq l$, then p is transmitted and the deficit counter is decremented by l: $d_i := d_i - l$.
- **Step 4.** The scheduler continues with the next queue. Steps 2–4 are repeated until there is a non-empty queue.

The DRR algorithm is not only scalable but can also easily be implemented in network routers. Beyond DRR, there are some other well-known fair schedulers like SFQ and WFQ. SFQ [74] does not allocate a queue for each flow, instead it uses a limited number of queues and distributes network traffic among these queues by a hash function. This stochastic approach enables efficient operation at the expense of less accurate approximation of the ideal fair scheduling. WFQ [73] allows network operators to define traffic classes and then assign different bandwidth proportions to them. Although WFQ is able to obtain fair allocation, it is difficult to implement in hardware due to its complexity.

Figure 4.13 shows the goodput performance of DFCP and TCP versions for the same test scenario presented in Section 4.1.1. Figure 4.13a indicates that, with DRR scheduler, DFCP can provide equal bandwidth sharing for the competing flows similar to the case

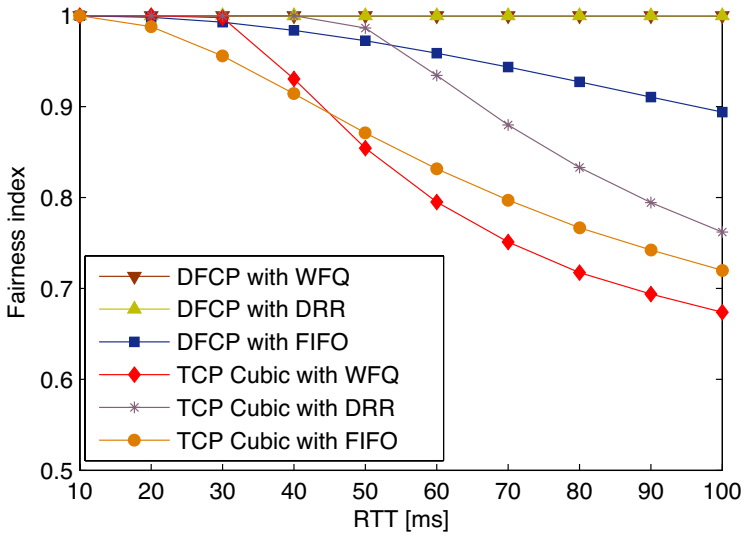


Figure 4.14. Intra-protocol fairness with WFQ, DRR and FIFO schedulers (simulation)

when WFQ was used. In addition, we can observe that applying DRR makes TCP less sensitive to the difference experienced between the RTTs of *flow 1* and *flow 2* resulting in better performance. Figure 4.13b demonstrates that the unfairness phenomenon is more pronounced if we use the simple FIFO scheduling algorithm. Even though, for both transport protocols the difference in goodput between the competing flows gets higher for increasing delay, for DFCP this change is much slower than for TCP Cubic.

Figure 4.14 presents a fairness comparison of DFCP and TCP Cubic using different schedulers. The results clearly show that DFCP can guarantee perfect fairness for the two competing flows independently of their RTTs if fair schedulers are used. Moreover, DFCP achieves better fairness than TCP with the much simpler FIFO algorithm as well, even if TCP is coupled with fair scheduling.

It is clear that in typical network conditions DFCP can obtain a higher degree of fairness compared to TCP for each queuing discipline. In other words, according to the results, current Internet architecture with FIFO queues would provide better fairness for competing flows by applying DFCP as a transport protocol instead of TCP. However, the highest degree of fairness can be realized by deploying DFCP together with fair schedulers, which can significantly improve TCP-based bandwidth sharing.

4.1.5 Scalability

On a typical bottleneck link hundreds of flows compete for the available bandwidth, and the capacity of these links is continuously increasing due to the development of communi-

Table 4.1. Performance scalability (simulation)

Bandwidth	**Normalized aggregate goodput (DFCP / TCP) [%]**		
	10 flows	*50 flows*	*100 flows*
0.1 Gbps	100 / 98	100 / 100	100 / 100
1 Gbps	100 / 96	100 / 98	100 / 99
10 Gbps	100 / 22	100 / 95	100 / 96

cation technologies. Good *scalability* is an important requirement for transport protocols meaning that they have to provide similar performance and fairness as the number of flows and the link capacity increase. The following simulations compare the scalability of two fundamentally different data transfer paradigms, DFCP with DRR scheduling and TCP Cubic with FIFO queue management. The results obtained for a 200 seconds long measurement period on the topology of Figure 3.11 with a 0.1 BDP buffer, and each flow experienced 100 ms of RTT.

Table 4.1 describes the performance scalability of the investigated transport protocols for different numbers of flows and link capacities. We computed the *normalized aggregate goodput* as the ratio of the aggregate goodput of concurrent flows and the maximum goodput can be achieved by a single flow. The normalized values are expressed in percentage and given for DFCP and TCP Cubic, respectively, separated by a slash mark. The results show that DFCP is able to gain the maximum performance irrespective of the number of flows and bottleneck bandwidth. In contrast, for TCP Cubic the normalized aggregate

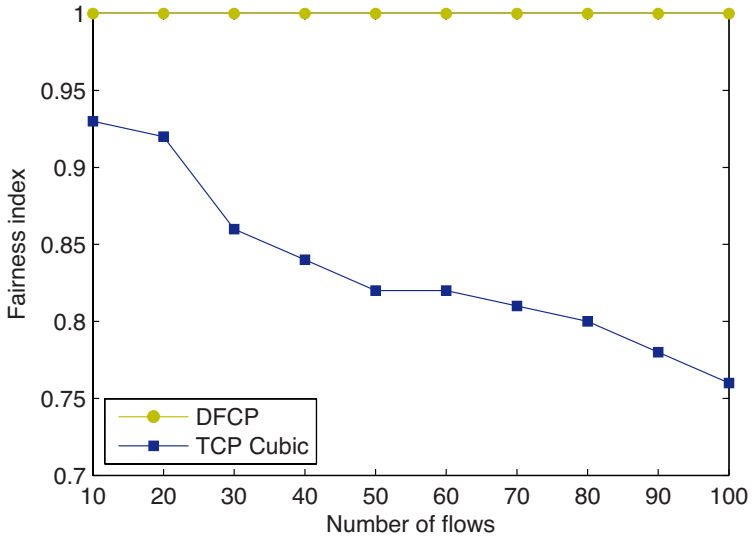


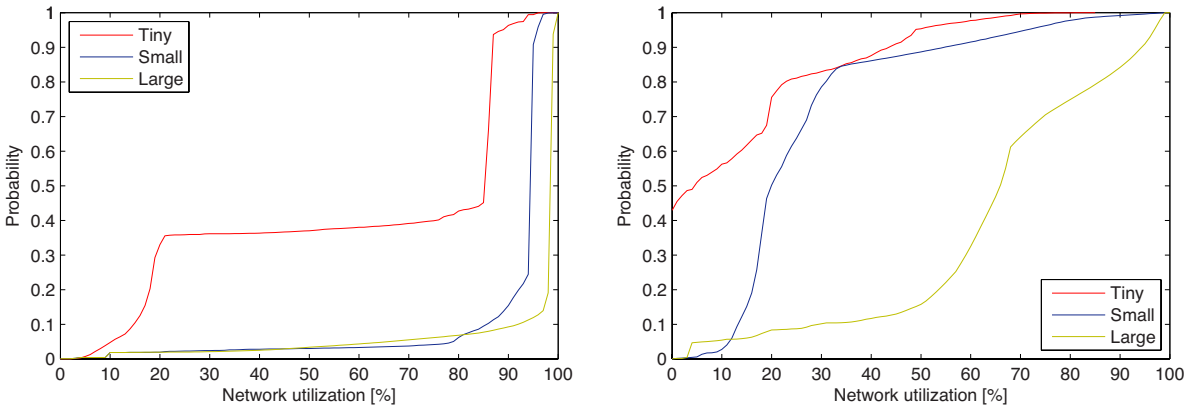
Figure 4.15. Fairness for increasing number of competing flows (simulation)

goodput increases with the number of flows, but decreases with the link capacity. For example, in the case of a 100 Mbps link the maximum performance can be obtained by 50 competing flows, however, an increase in the link capacity by two orders of magnitude leads to a 5% performance degradation. Moreover, high capacity links cannot be fully utilized by a small number of flows since the round-trip time limits the transmission rate of individual flows. In this special case, 100 ms of RTT results in a goodput reduced to approx. 200 Mbps for each flow (see Figure 4.2), and hence the underutilization of the 10 Gbps link by 10 flows.

Figure 4.15 demonstrates the fairness scalability of DFCP and TCP Cubic for increasing number of flows. In this scenario, each flow experienced the same delay to avoid the phenomenon of RTT unfairness. In spite of that the tendency is obvious for TCP Cubic: the larger the number of concurrent flows, the lower the fairness index. However, in contrast to all of these results DFCP can ensure fair bandwidth sharing independently of the number of competing flows.

4.1.6 Network Utilization

Most network operators choose over-provisioning as a way to satisfy resource demands even during the busy hours of the day [90], which is based on the assumption that quality issues can be readily addressed by allowing some excess capacity. However, this approach can often be very inefficient leading to degraded QoE [91], and for some environments, it is simply not a viable option due to various constraints. *Utilization* is a key measure to support the proper design and sizing of networks. The following results reveal the extent



(a) DFCP with DRR (b) TCP with FIFO

Figure 4.16. CDF of network utilization for different buffer sizes (simulation)

Table 4.2. The ratio of load levels for different buffer sizes (simulation)

Buffer size	**Underloaded**		**Normal**		**Overloaded**	
	DFCP	*TCP*	*DFCP*	*TCP*	*DFCP*	*TCP*
Tiny	0%	68%	10%	32%	90%	0%
Small	0%	45%	6%	52%	94%	3%
Large	0%	0%	5%	73%	95%	27%
Average	0%	38%	7%	52%	93%	10%

of utilization can be achieved by different transport mechanisms. In these experiments, DFCP with DRR scheduling and TCP Cubic with FIFO queue management are investigated, as well as network utilization is interpreted as the total mean utilization of all bottleneck links and buffers. Simulations were conducted on the parking lot topology of Figure 3.12 with bottleneck link capacities 1 Gbps and 400 Mbps.

Figure 4.16 shows the cumulative distribution function (CDF) of network utilization for different buffer sizes. We define three categories called *tiny*, *small* and *large* corresponding to buffer space of 0.01 BDP, 0.1 BDP and 1 BDP, respectively. The evaluation results indicate that, with larger buffers higher utilization can be achieved for both paradigms, but DFCP surpasses TCP irrespective of what type of buffers is used. We can also see that DFCP can utilize more than 90% of network resources in most cases, both with small and large buffers. The difference in utilization levels is the most pronounced between tiny and small buffers. In contrast, the network utilization significantly drops for TCP not only when tiny buffers are used but also when small ones. Overall, we can say that, regarding resource utilization, TCP can bring out the maximum with large buffers while DFCP is able to do so even with small buffers.

Table 4.2 gives the ratio of different load levels using tiny, small and large bottleneck buffers. *Underloaded*, *normal* and *overloaded* regimes are defined as an operational state with mean network utilization falling in the interval of 0–20%, 20–80% and 80–100%, respectively. The results point out that DFCP never keep the network underutilized, and in about 90% of the time it works in the overloaded regime with highly saturated links and buffers. Regarding TCP, the case is the opposite: in most of the time, the network operates in the underloaded or normal regimes. The results also reveal that, with large buffers, the network does not get underloaded whereas the use of tiny buffers prevent the network from overloading.

4.2 Conclusion

In this chapter, we carried out a steady-state analysis of two alternative paradigms as possible data transport mechanisms for future networks: the digital fountain based DFCP and the congestion control based TCP. In order to draw solid conclusions, we studied the operation of these protocols on various network topologies and multiple platforms including our laboratory testbed, the Emulab network emulation environment and the ns-2 network simulator. We showed that the goodput performance of DFCP is significantly better than in the case of the investigated TCP versions in a wide range of packet loss rates and round-trip times. The results also pointed out that DFCP is able to obtain maximum performance even with small buffers, which could make it attractive for all-optical networks. Moreover, DFCP provides fair bandwidth sharing among competing flows independently of their RTTs. Although perfect fairness can only be achieved when fair schedulers (e.g. DRR) are used, DFCP can ensure better fairness than TCP even in the absence of any fair scheduler and if the simplest FIFO algorithm is applied. Finally, digital fountain based transport guarantees good scalability and stability, both in terms of performance and fairness for increasing number of flows and link capacity while keeping the network highly utilized. The results suggest that it is a promising approach with numerous beneficial properties and a broad spectrum of possible applications.

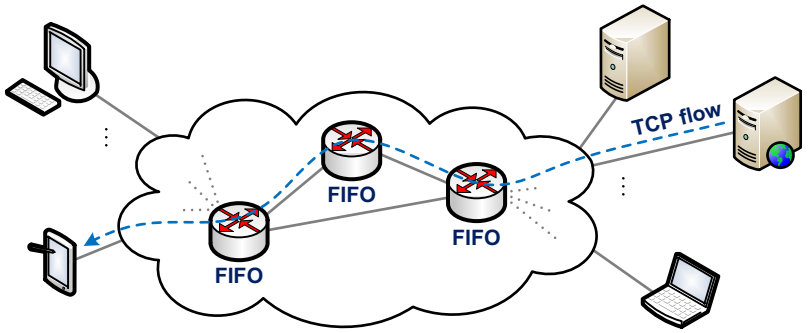
Chapter 5

Characterization of Transport Mechanisms under Dynamic Traffic Conditions

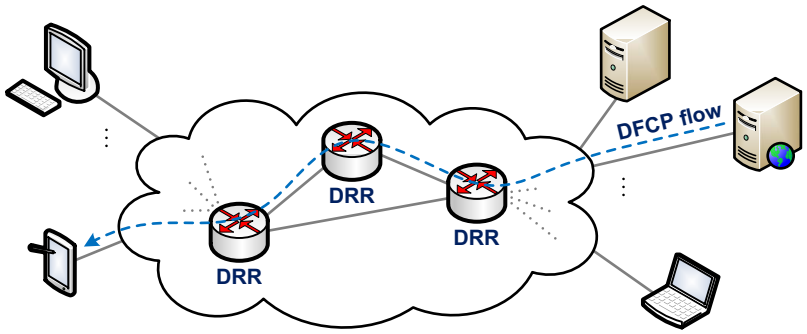
The volume of Internet traffic has been growing exponentially in the last years, and this trend is expected to intensify in the future [92]. In typical backbone networks hundreds of thousands of flows, originated from different users and versatile applications, compete for the available resources. Such a heterogeneous mixture of traffic flows leads to a continuously changing environment, hence it is crucial to deeply understand the behavior of the underlying data transfer mechanisms regarding many features like stability, convergence and responsiveness. In this chapter, we study the characteristics of digital fountain based transport (discussed in Chapter 3) under dynamic network conditions, and carry out a comparison with the traditional approach relying on TCP's congestion control.

5.1 Background

Since the early days of the Internet, congestion control introduced by TCP, has played the key role in reliable host-to-host communication. In the last decades, the characteristics of network traffic have changed considerably due to the evolving technologies and the diversity of applications. Today's Internet is a large-scale, highly dynamic network in which sudden variations are common due to topology and bandwidth changes. While a great portion of TCP evaluation studies deal with performance analysis solely in static environments, some researchers emphasize the importance of exploring how responsive a transport protocol is under rapidly changing conditions [93, 94]. In spite of the significant research efforts devoted to optimize the operation of TCP for a wide range of network



(a) Congestion Control based Architecture



(b) Digital Fountain based Architecture

Figure 5.1. Network architectures relying on different transport mechanisms

environments, it seems that congestion control may not be able to cope with the increasing demands of future networks. The comprehensive performance evaluation of our digital fountain based transport paradigm (see Chapter 4) revealed that this new approach has several potential benefits. In this chapter, we deeply investigate the *dynamic behavior* of our proposal, and carry out a comparison with the traditional TCP-based solution of current Internet.

Nowadays, most network routers apply a simple FIFO queue management algorithm to handle packet buffering. By using this method, when the queue becomes full, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic. Due to the fact that network traffic consists of thousands of competing flows, a data transfer paradigm has to ensure fair bandwidth sharing. In the TCP-based architecture fairness is managed at the host side, but different TCP versions realize different types of fairness [71]. In this analysis, the operation of TCP Cubic with FIFO queue management is investigated (see Figure 5.1a), and we call this approach as *Congestion Control based Architecture (CCA)*. Chapter 3 and Chapter 4 highlighted that our digital fountain based

approach can overcome TCP in terms of high loss and delay tolerance, fair bandwidth allocation, low buffer space demand and fast completion of traffic flows. Here the operation of DFCP with DRR scheduling is evaluated under dynamic network conditions (see Figure 5.1b), and we refer to this concept as *Digital Fountain based Architecture (DFA)*.

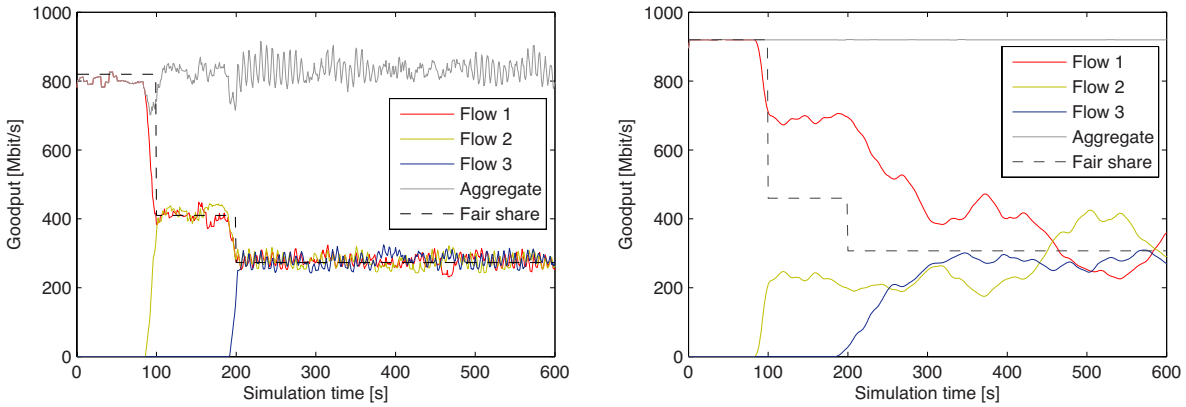
5.2 Dynamic Behavior Analysis

The experiments were performed on a dumbbell topology with N senders and receivers as shown in Figure 3.11. In the measurement scenarios we examined real-world situations, which typically occur in a dynamic environment by varying the link capacity and delay, the buffer size and the number of competing flows. Simulations lasted for 600 seconds, and if not mentioned otherwise, the bottleneck link capacity was set to 1 Gbps, the round-trip time was fixed at 50 ms and the buffer size (denoted by b) was equal to the bandwidth-delay product. In DFCP, the redundancy and the window size parameters were adjusted to $\epsilon = 0.05$ and 1000 blocks, respectively.

5.2.1 Stability and Convergence

Network traffic is generated by heterogeneous applications that results in many concurrent flows traversing different network paths with multiple bottlenecks from source to destination. The transmission rates of these flows fluctuate rapidly since the currently used congestion control based transfer mechanism could not adapt to changing conditions as fast as needed. *Stability* is an important property from both traffic engineering and user experience points of view [70]. Rate variations often lead to the oscillation of queue length that can eventually cause buffer overflows. Such an undesirable behavior can result in the loss of synchronization among competing flows, periodic underutilization of link capacity and degraded quality of service. It is also crucial regarding efficiency how fast a flow can obtain its equilibrium rate or *converge* to the fair share in a dynamic environment [70].

In Figure 5.2, the dynamics of three concurrent flows is illustrated when they were started with different delays, namely at 0, 100 and 200 seconds. The goodput gives the current useful data transmission speed in one second resolution, and the curves were smoothed by using a 10 seconds long moving window. We can observe that, for CCA, flows converge slowly to the fair share and then their goodput highly fluctuates around it. In the case of DFA, the convergence time is very low whereas the fluctuation around the fair share remains moderate. However, the transfer mechanism of DFA leads to a more



(a) Digital Fountain based Architecture (b) Congestion Control based Architecture

Figure 5.2. Dynamics of concurrent flows started with different delays and their convergence to the fair share

bursty transmission than that of CCA, which is due to the trade-off between the window size and the burstiness of traffic.

To describe the transient fairness of transport mechanisms, we measured the *goodput ratio* of two competing flows started with unequal shares of the bottleneck bandwidth for DFA and CCA, respectively. Figure 5.3 shows a scenario when the flows were launched at 0 and 100 seconds, hence *flow 1* had been utilizing the total available bandwidth at the time of starting the second flow. The vertical axis gives the goodput ratio of *flow 2* to *flow 1*. We can see that, for CCA, transmission rates of competing flows converge slowly, but they remain stable at different time scales. In contrast, for DFA, while the goodput convergence is fast providing high degree of long-term fairness, a slight oscillation around the equal share (i.e. the goodput ratio of 1) can be observed at small time scales.

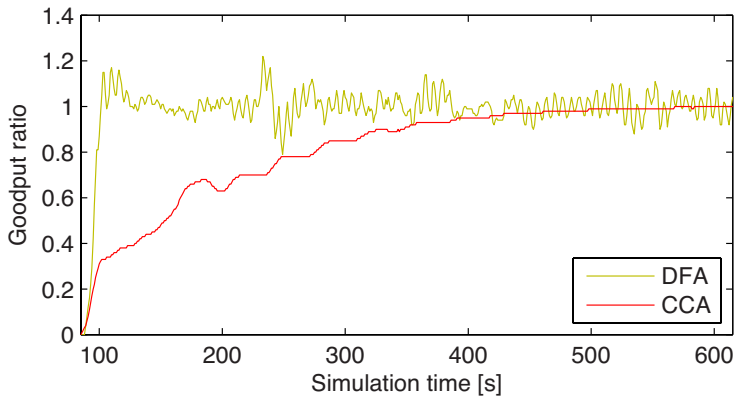


Figure 5.3. Goodput ratio in the function of time for two delayed flows

A frequently used metric to quantify the convergence speed is *δ-fair convergence time* [93], which can be given as the time taken by two flows to obtain a bandwidth allocation of $(\frac{1+\delta}{2}B, \frac{1-\delta}{2}B)$ starting from $(B - B_0, B_0)$ where $B >> B_0$. Using the settings of $B = 1000$ and $B_0 = 0$, the average δ-fair convergence times provided by DFA and CCA are summarized in Table 5.1. These results have great significance in the case of short downloads and suggest that CCA cannot guarantee reasonable fairness for many Internet applications since about half of network flows last less than a few seconds.

Table 5.1. Convergence time to the fair share

Paradigm	**δ-fair convergence time**		
	$\delta = 0.1$	$\delta = 0.3$	$\delta = 0.5$
DFA	2 sec	1 sec	1 sec
CCA	180 sec	60 sec	5 sec

5.2.2 Responsiveness

One of the key concerns in the design of transport protocols is the ability to handle abrupt change of network parameters and traffic conditions. In a real network competing flows governed by different transfer mechanisms often face with quick variations mainly originated from routing and bandwidth changes, or sudden congestion. *Responsiveness* is of high importance describing how fast and accurately a transport protocol can adapt to these environmental factors [70].

In this section, we focus on the change of the available bandwidth and quantify the responsiveness of per-flow and aggregate traffic for the two different data transfer paradigms. To this end, we defined and calculated a metric called *adaptation error* as follows. Let g_i be the goodput of flow i and f_i the ideal fair share for flow i taking into account the bandwidth change pattern. Using these notations the adaptation error of per-flow (e_p) and aggregate traffic (e_a) can be computed by the following formulas:

$$e_p = \frac{1}{n}\sum_{i=1}^{n}\frac{|g_i - f_i|}{f_i} \quad \text{and} \quad e_a = \frac{\sum_{i=1}^{n}|g_i - f_i|}{\sum_{i=1}^{n} f_i} \tag{5.1}$$

where n denotes the number of flows competing for the bottleneck bandwidth.

We analyzed the behavior of 10 competing flows by periodically halving the available bandwidth of the bottleneck link. Specifically, the bandwidth was reduced from 1000 Mbps to 500 Mbps in the interval of [100,200], [300,340], [380,420] and [460,500] as illustrated in

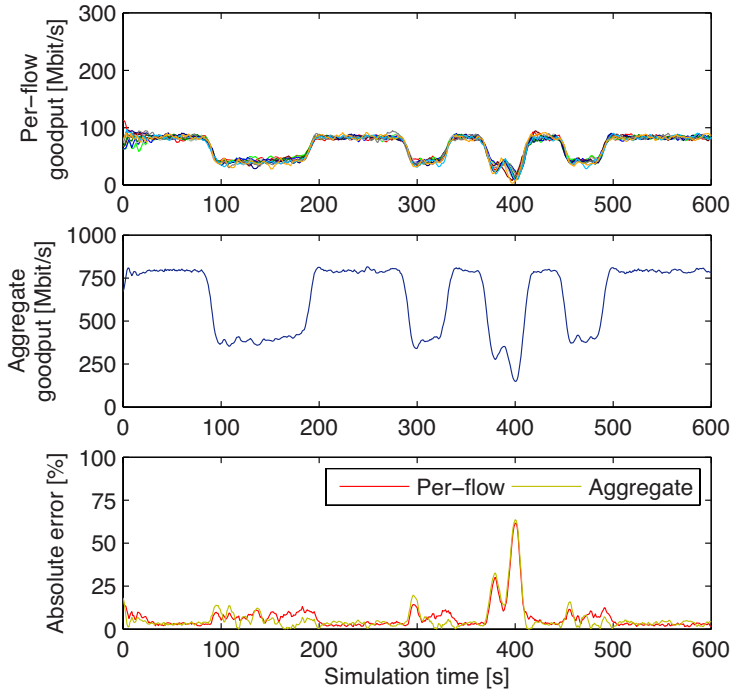


Figure 5.4. Responsiveness of per-flow *(top)* and aggregate *(middle)* traffic, and the adaptation error *(bottom)* for DFA with a buffer size of 100 packets

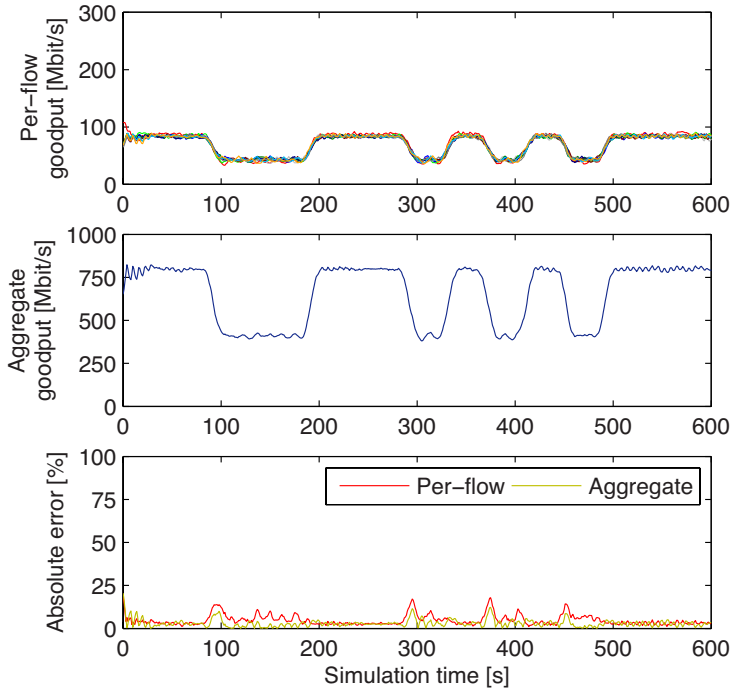


Figure 5.5. Responsiveness of per-flow *(top)* and aggregate *(middle)* traffic, and the adaptation error *(bottom)* for DFA with a buffer size of 5000 packets

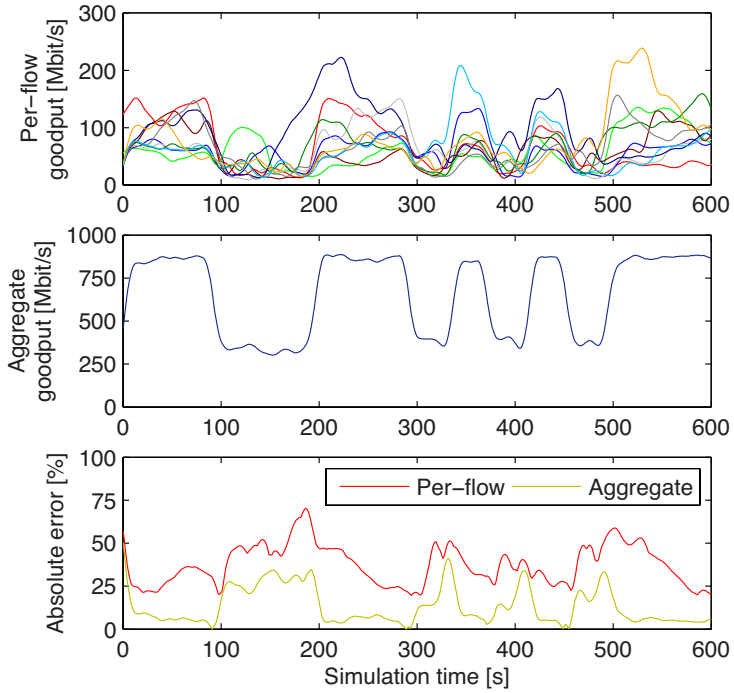


Figure 5.6. Responsiveness of per-flow *(top)* and aggregate *(middle)* traffic, and the adaptation error *(bottom)* for CCA with a buffer size of 100 packets

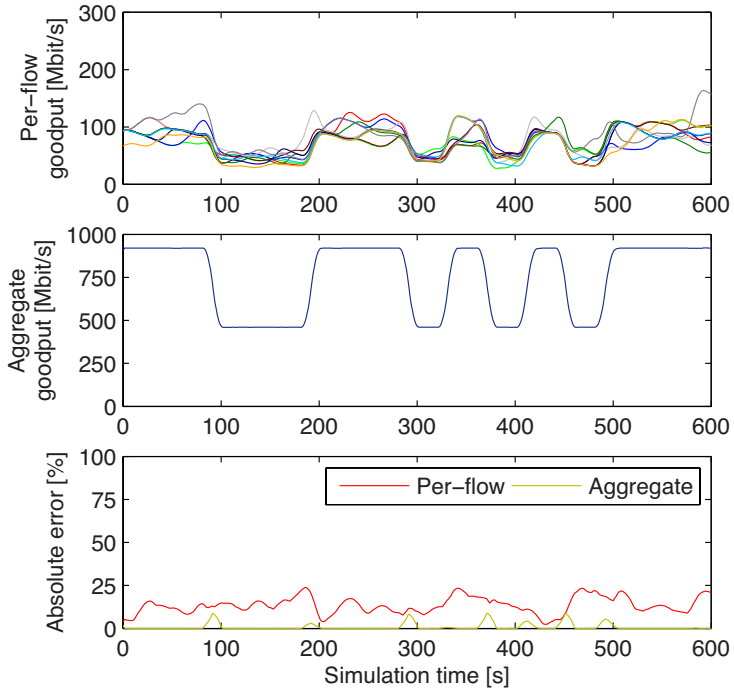
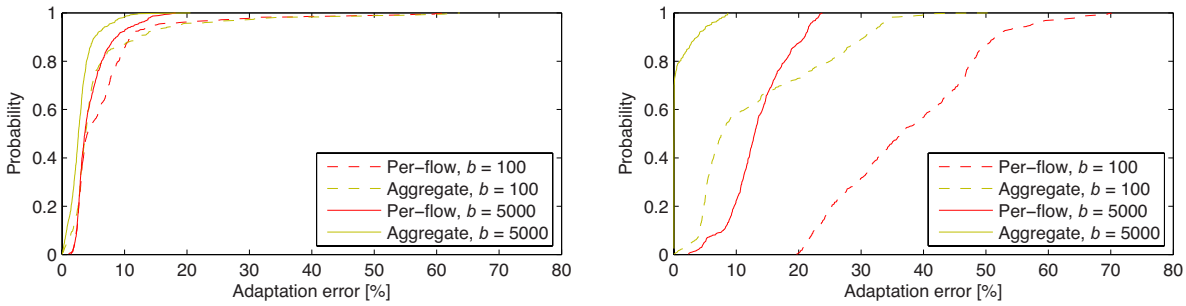


Figure 5.7. Responsiveness of per-flow *(top)* and aggregate *(middle)* traffic, and the adaptation error *(bottom)* for CCA with a buffer size of 5000 packets



(a) Digital Fountain based Architecture (b) Congestion Control based Architecture

Figure 5.8. CDF of adaptation error of per-flow and aggregate traffic for DFA and CCA

Figure 5.9. In Figure 5.4–5.7, the responsiveness of per-flow and aggregate traffic, and the adaptation error are shown for DFA and CCA. Figure 5.8 depicts the CDF of error whereas Table 5.2 summarizes the mean and standard deviation of values. To reveal the adaptation capability of transfer paradigms to different network environments, we investigated both small ($b = 100$) and large ($b = 5000$) buffer sizes (measured in packets). The good operability with small buffers is a mandatory requirement for all-optical communication and also makes it possible to avoid the bufferbloat phenomenon experienced in current Internet mainly due to the use of over-sized router memories [91].

Regarding the small buffer case ($b = 100$), we can see that the adaptation speed of CCA flows to bandwidth changes is very low (Figure 5.6). Ideally, each flow would receive an equal share of the bottleneck link, but in this case some flows react too aggressively and some too mildly to changing network conditions. This behavior leads to uneven bandwidth allocation especially during the periods after the available bandwidth is doubled. For example, in the interval of [200,300] the maximum perceived difference in goodput between individual flows exceeds 200 Mbps, which is two times more than the fair share. In spite of the high unresponsiveness of single flows, the aggregate traffic roughly follow the change

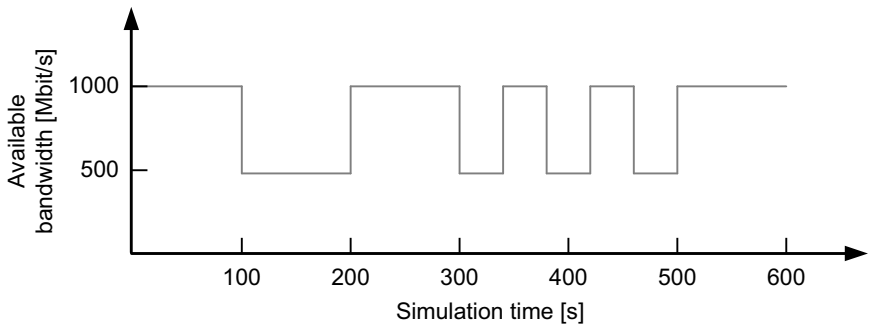


Figure 5.9. The change pattern of the available bandwidth

Table 5.2. The mean *(left)* and standard deviation *(right)* of the adaptation error in percentage

Source	DFA				CCA			
	$b = 100$		$b = 5000$		$b = 100$		$b = 5000$	
Single flow	2.7	2.6	2.3	1.9	26.1	20.7	2.4	4.5
10 flows, per-flow	6.7	9.2	4.8	4.3	37.5	23.7	13.5	10.4
10 flows, aggregate	6.1	8.4	3.0	2.1	13.2	10.4	0.9	1.9

pattern. The per-flow adaptation error is significant and ranges between 20% and 70% with a mean of 38% (Table 5.2) while in the case of the aggregate more than half of the samples are below 10% (Figure 5.8) with a mean of 13%. For DFA, we can experience a moderate oscillation of per-flow goodput around the fair share (Figure 5.4), however, the stability of aggregate traffic does not show noticeable difference compared to that of CCA. Apart from some outlier values, the error rate remains moderate with an average of 7% and 6%, hence it is only slightly higher for per-flow traffic. If large buffers with size close to the BDP ($b = 5000$) are used in CCA routers (Figure 5.7), individual flows can follow much more smoothly the bandwidth changes, which results in high responsiveness of the aggregate traffic. While the per-flow adaptation error does not exceed 25%, for the aggregate traffic, the error rate is negligible and can only be measured when changes occur. Although the use of large buffers also has a positive effect on the adaptation accuracy of DFA, the improvement is barely noticeable as can be seen in Figure 5.8. To sum up, our measurements indicate that CCA can provide better adaptivity than DFA only in the case of the aggregate traffic and if sufficiently large buffers are applied.

5.2.3 Saturation Time

The operation of congestion control algorithms consists of two main transmission phases. In the initial phase, TCP gradually increases the sending rate until the bottleneck buffer is filled. Then, it is followed by an equilibrium state when the protocol achieves the maximum transmission rate and tries to keep it stable. The length of the transient phase highly determines the download efficiency of short-lived flows, therefore it can affect the quality of experience for many applications. In order to capture this behavior, we defined a performance metric called *saturation time* [10], which can be given for a loss-based protocol as the time elapsed from the starting of a flow until the first packet is dropped. Queue saturation time (QST) is a good indicator of how fast a transport protocol can obtain its steady-state performance.

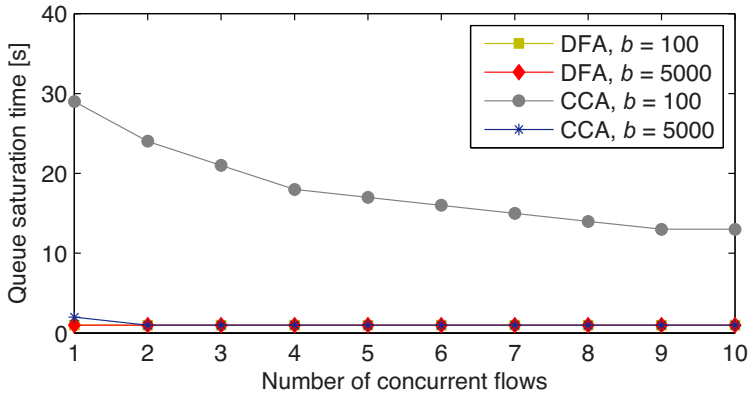


Figure 5.10. Queue saturation time for increasing number of flows

Figure 5.10 shows the queue saturation time can be provided by the two data transmission paradigms for increasing number of flows. Similarly to responsiveness we investigated the impact of both small and large buffers. If we use a buffer size greater than the bandwidth-delay product, the bottleneck queue is saturated in a very short time and independently of the number of concurrent flows for both transfer mechanisms. The figure also clearly demonstrates that, by using a buffer size of only 100 packets, QST becomes dramatically high for CCA and it decreases when many concurrent flows share the bandwidth of the bottleneck link. However, even if the results suggest that we can theoretically achieve low QST values for hundreds of competing flows, CCA is unable to handle such amount of network traffic with small buffers in comparison to DFA.

In Figure 5.11, the queue saturation time is depicted for different round-trip time values. We can see that, using CCA, the round-trip time considerably affects QST in the case of a particular buffer size. With a buffer size of 100 packets QST is already noticeable

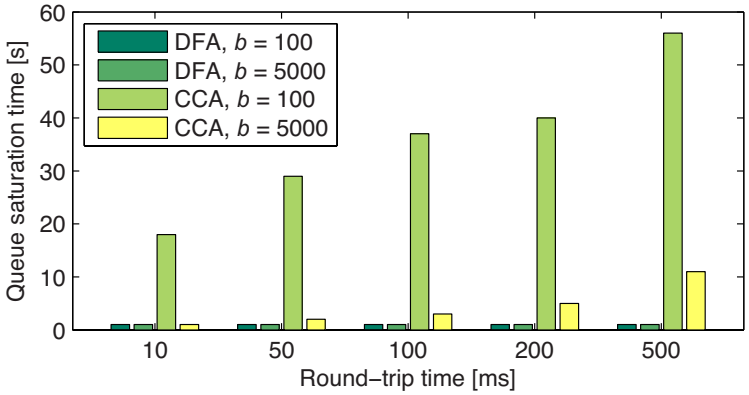


Figure 5.11. Queue saturation time for different round-trip times

on low-latency links and it increases for higher RTTs. The same tendency can be observed with a larger buffer of 5000 packets, but the increase in QST is less significant. In contrast, DFA is able to keep QST low even in high-delay environments.

5.3 Conclusion

In this chapter, we investigated the dynamic behavior of our DFCP-based data transfer paradigm (DFA) and compared it to the current Internet architecture (CCA) relying on the congestion control mechanism of TCP. The simulation results revealed that, while CCA can work with moderate goodput oscillation at small time scales, DFA is more stable in the long run and can guarantee fast convergence for competing traffic flows. We also found that DFA is able to cope with sudden change of network conditions regarding both per-flow and aggregate traffic independently of the buffer size. CCA shows better adaptivity only in the case of aggregate traffic and if sufficiently large buffers are used, with small router memories CCA is highly unresponsive. Furthermore, DFA provides low queue saturation time making QoE improvement possible for many applications.

Chapter 6

Available Bandwidth Estimation in Mobile Networks

In the recent years, a significant research effort has been devoted to the development of bandwidth estimation techniques and tools due to the broad range of possible applications. The vast majority of bandwidth estimation algorithms are designed and optimized for wired networks. Therefore, these solutions not only provide inaccurate results in wireless environments but also rely on some information usually not known in advance, or produce a severe additional load on the network. Specifically, in mobile networks the continuously varying characteristics of radio links make it an extreme challenge to estimate the currently available bandwidth. In this chapter, we present a bandwidth estimation method worked out for mobile networks, which models the dynamics of the bottleneck queue and identifies its busy periods. Our algorithm can estimate the unused bandwidth by exploiting the user-generated downlink network traffic with negligible extra load. The operation of the algorithm is demonstrated on real traffic traces captured by a mobile device in a 3G network.

6.1 Background

Traditionally, bandwidth is used as a measure quantifying the data transfer rate that a network link or path can provide. However, it is important to distinguish between the different meanings of the term *bandwidth*. In the literature, there are three frequently used interpretations [95]: the *maximum possible bandwidth* that a link or path can deliver (capacity), the *maximum unused bandwidth* at a link or path (available bandwidth), and the *maximum throughput* can be obtained by a single TCP connection (bulk transfer capacity).

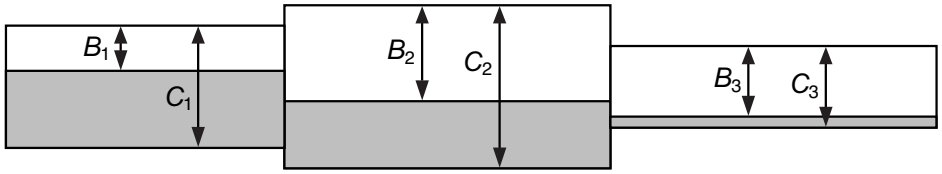


Figure 6.1. The tight and narrow link of a network path

The available bandwidth (ABw) of a link is described as the unused bandwidth of that link for a given time interval [95]. The ABw is a time-varying metric since it depends on the traffic load, also known as cross-traffic. The average utilization of the link can be given as follows:

$$u^{\tau}(t) = \frac{1}{\tau}\int_{t-\tau}^{t} u(x)dx \tag{6.1}$$

where $u(x)$ is the instantaneous utilization of the link at time x with values falling between 0 and 1, as well as τ denotes the averaging interval. Let C_i and u_i be the capacity and the mean utilization of link i, respectively. In this case, the ABw is defined as

$$B_i^{\tau}(t) = (1 - u_i^{\tau}(t))C_i. \tag{6.2}$$

If we have a path with n links, the end-to-end ABw can be given as the minimum of all values, that is

$$B^{\tau}(t) = \min_{i=1,2,\ldots,n}\{B_i^{\tau}(t)\} = \min_{i=1,2,\ldots,n}\{(1 - u_i^{\tau}(t))C_i\}. \tag{6.3}$$

The link with the minimum ABw is called the *tight link* of the path, while the link having the lowest capacity is the *narrow link*. Figure 6.1 shows a simple network model consisting of three consecutive links where the used bandwidth is colored gray and the rest of the area corresponds to the ABw. One can see that the first link has the lowest ABw, whereas the link with the lowest capacity is the third one. This example illustrates that the tight link and the narrow link can be different along a network path.

In the last decade, a plenty of bandwidth estimation algorithms and tools have been developed in order to meet the increasing demands [95, 96]. The design of efficient bandwidth estimation methods is not easy because some contradicting requirements are need to be fulfilled. An ideal algorithm would provide high estimation accuracy, fast operation

and low overhead. However, in practice not all these features are equally relevant, and it highly depends on the application area which ones have to be optimized. For example, in the case of transport protocols low overhead and low estimation time are required, but they do not need high accuracy for proper operation, a rough estimate is acceptable [96].

Available bandwidth estimation is one of the most challenging tasks in the context of bandwidth estimation methods addressed in many papers [95, 97]. The majority of ABw estimation techniques send probe packets to the receiver utilized in the estimation process and are based on two basic models: the Probe Gap Model (PGM) and the Probe Rate Model (PRM). PGM exploits the information about gap dispersion between two consecutive probe packets at the receiver. The gap dispersion has a strong correlation with the amount of cross-traffic in the tight link, that is, with the link having the lowest available bandwidth. The methods using PGM (e.g. Abing, Spruce) first determine the amount of cross-traffic, and then subtract the result from the known capacity of the tight link. PRM tools (e.g. Pathload, pathChirp, DietTopp) are based on the idea of self-induced congestion where probe packets are sent at increasing rates to the receiver, and the available bandwidth is determined by studying the change of the queuing delay and measuring the output rate.

The issue of estimating the bulk transfer capacity (BTC) is investigated only in a few papers. BTC is defined as the maximum throughput can be obtained by a single TCP connection [98]. For example, Allman introduces a BTC measurement tool in [99] and presents its empirical evaluation together with the investigation of reliability. Gardner et al. propose a novel method for estimating the BTC of an IPv6 network path, conducted from a single point to a non-instrumented target [100]. In fact, BTC is very hard to measure since it can be affected by several factors such as the type of cross-traffic, the number of competing TCP connections, the buffer space in routers and the queuing policies [95].

Unfortunately, the vast majority of bandwidth estimation tools discussed above are designed for wired networks, therefore they cannot provide accurate results and short convergence time in wireless environments, especially in mobile networks. Cellular networks bring a lot of additional challenges, which make bandwidth estimation more difficult compared to wired networks. The bandwidth available by the user is continuously varying due to the changing network conditions such as the location and motion speed of the mobile device, the number of users in the current cell, the signal strength, handovers and many other effects [101]. Negreira et al. discuss the issues of end-to-end measurements over GPRS-EDGE networks in [102] and presents a new methodology capable of providing ac-

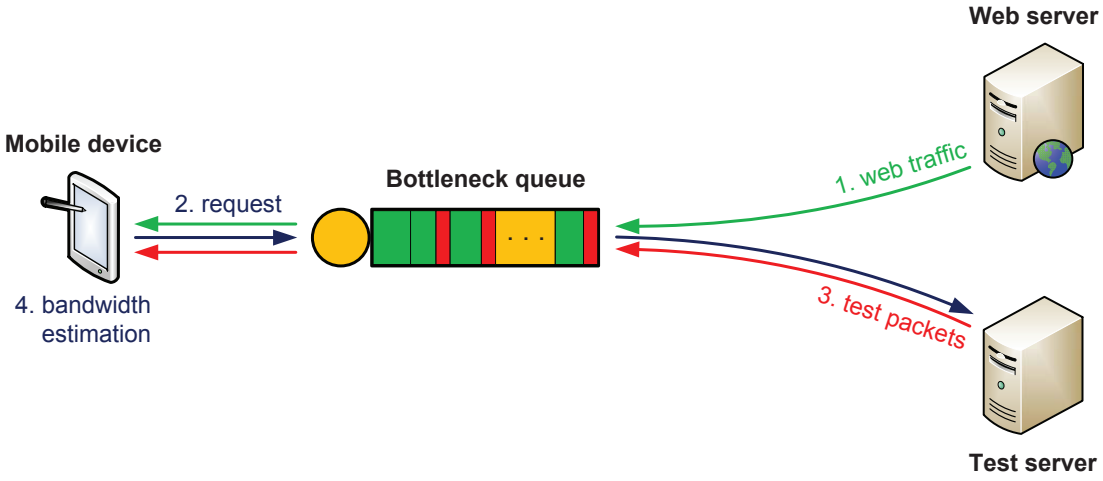


Figure 6.2. The operation of the bandwidth estimation scheme

ceptable results in such environments. The authors of [103] defined a so-called in-context network performance measure to express the user experience when they are interacting with their mobile devices. They carried out a large-scale measurement study using data collected across cell subscribers and controlled experiments. They pointed out that, to obtain accurate results, performance measurements must be conducted on devices, which are actively used during the measurement time frame, currently exchanging limited user traffic and can be found in the same position and environment since the last usage of device. Bergfeldt et al. performed an evaluation of bandwidth measurement tools over a high-speed downlink UMTS channel by running experiments in a commercial mobile network [104]. In this work, they investigated several test scenarios by using various types of cross-traffic and bottlenecks. The results showed that algorithms can significantly under- or overestimate the available bandwidth under certain network conditions.

6.2 Bandwidth Estimation Scheme

6.2.1 Basic Idea

Our bandwidth estimation method is designed to estimate the *unused bandwidth* available on a mobile device by exploiting the user-generated downlink network traffic with negligible extra load. Figure 6.2 shows the main concept in an architectural view with the main hardware and software components.

The bandwidth estimation scheme works as follows. The downlink network traffic generated by the user is continuously monitored on the mobile device. When the type

and amount of traffic are considered as sufficient to initiate the bandwidth estimation process, the mobile device sends a request signal to the test server. The test server starts to generate a sequence of test packets with a specified frequency and sends it towards the mobile device. The generation of test packets at the test server is finished if a stop request is received from the mobile device or a timeout is occurred. By this way, test packets are injected into the user-generated downlink traffic. Our algorithm running on the mobile device simply estimates the unused bandwidth by dividing the amount of traffic observed between two test packets by the elapsed time. However, it can be highly inaccurate, therefore the key step is to determine which periods of the data flow are eligible for performing estimation. The algorithm utilizes two basic information to achieve this: (1) the fixed test packet generation interval and (2) the measured test packet inter-arrival times (IAT). Based on these information, our method models the queue dynamics of the bottleneck link (assumed to be the wireless connection between the base station and the mobile device) and identify its busy periods in order to enhance the estimation accuracy.

In summary, the presented available bandwidth estimation algorithm has the following capabilities:

- no specific information about the network is needed (e.g. bottleneck link capacity);
- the bandwidth estimation algorithm runs only when the user is active (e.g. browsing the web), and a probe traffic is injected into the user-generated downlink network traffic;
- since the probe traffic consists of a sequence of small-sized packets, the estimation scheme causes a very low additional network load;
- by modeling the dynamics of the bottleneck queue and identifying the busy periods, it can provide reasonable accuracy in spite of quick and high variations often seen in mobile data networks.

6.2.2 Algorithm Description

The most challenging task is to capture the busy periods of the bottleneck queue. In other words, we find those intervals in the downlink traffic trace when the queue is not empty, and hence, enqueued packets will be serviced at maximal rate. The pseudocode of the algorithm can be seen in Algorithm 1 where d, th, and gap denote the delay between the generation of test packets, the positive threshold used for busy period detection and the

highest IAT can be accepted in the queue modeling phase, respectively. Furthermore, t_i is the arrival time of the i^{th} test packet captured at the mobile device, t_{i+1} is the arrival time of the $(i+1)^{th}$ test packet and n is the number of test packets in the traffic trace. The boolean variables m and b indicate if the queue modeling and busy period detection phases are active.

Algorithm 1: Available bandwidth estimation

input : $trace, d, th, gap$
output: bw

$m \leftarrow false$; $b \leftarrow false$;
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $t_{i+1} - t_i = d$ **and** $m = false$ **then**
    $q \leftarrow 0$;
    $m \leftarrow true$;
  else if $t_{i+1} - t_i > gap$ **and** $m = true$ **then**
    $m \leftarrow false$;
    $b \leftarrow false$;
  else if $m = true$ **then**
    $q \leftarrow t_{i+1} - t_i - d + q$;
    if $q \geq th$ **and** $b = false$ **then**
      $s \leftarrow t_i$;
      $b \leftarrow true$;
    else if $q < th$ **and** $b = true$ **then**
      $rates \leftarrow \texttt{Add}\left(\frac{\text{amount of traffic in } [s,t_i]}{t_i - s}\right)$;
      $b \leftarrow false$;
    end
  end
end
return $bw \leftarrow \texttt{Mean}(rates)$;

In the following, we discuss each main step performed by our bandwidth estimation scheme, which is also illustrated in the flow chart of Figure 6.3:

- **Step 1** ***(initialization).*** As a first step, the algorithm detects whether the queue is empty by finding an IAT of two successive test packet arrivals, which is equal to the test packet generation interval ($t_i - t_{i-1} = d$). In ideal case, it means that the i^{th} test packet will experience zero waiting time ($q_i = 0$).

- **Step 2** ***(queue dynamics modeling).*** The algorithm starts to model the queue dynamics and computes the current waiting time by determining the difference

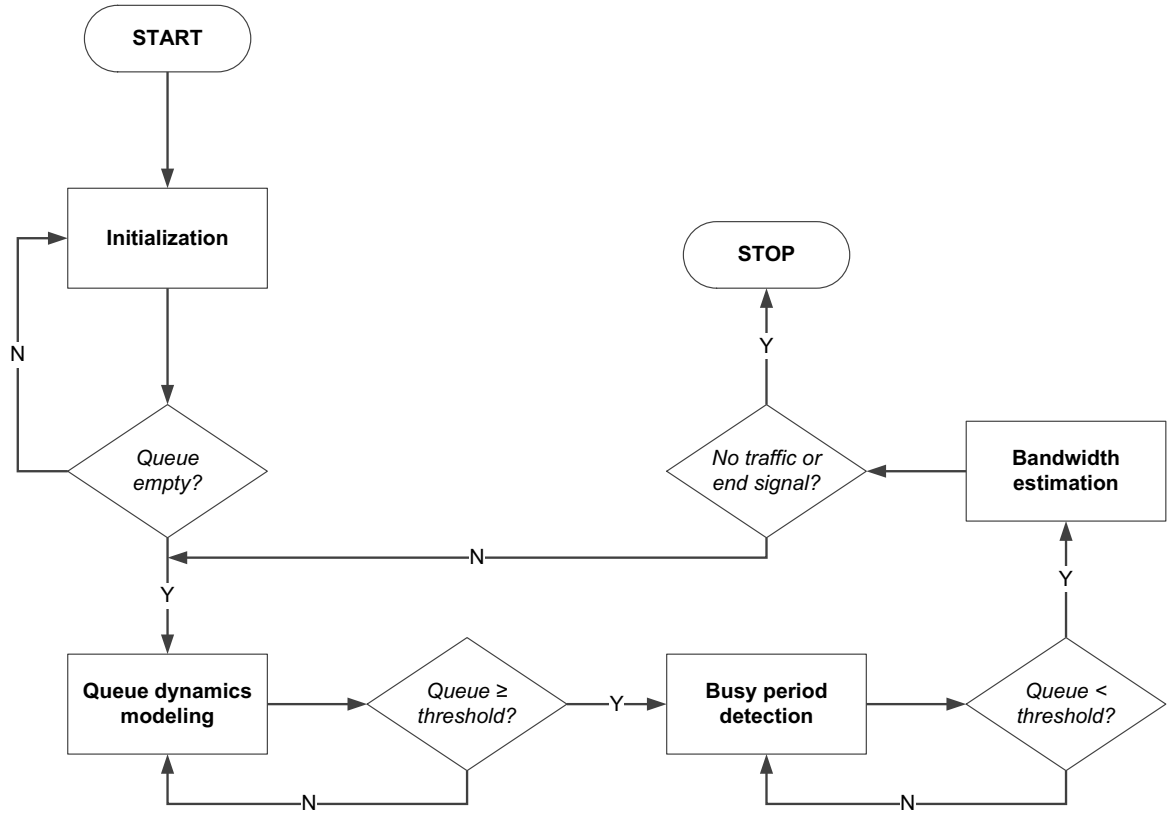


Figure 6.3. The flow chart of the operation phases

between the IAT ($t_i - t_{i-1}$) and the generation interval (d). In the following time slots, current waiting time comes from the sum of this difference and the waiting time calculated in the previous slot ($q_i = t_i - t_{i-1} - d + q_{i-1}$). In theory, the result should be a non-negative value, but in practice there are some factors (e.g. jitter), which can turn it to negative.

- **Step 3** ***(busy period detection).*** Ideally, when the cumulative waiting time (referred to as queue length) becomes greater than zero, we could say that the queue is busy. However, to make the detection more accurate in realistic environments, the algorithm uses a positive threshold (th) instead of zero as a reference point to identify the start of the busy period (see Figure 6.4). If the waiting time drops below this threshold, it indicates the end of the busy period. Another effect which can lead to the termination of the busy period detection and the queue modeling phase is observing a high test packet IAT probably not due to the impact of user-generated traffic. We call these IATs as outliers, and the outlier detection can also be con-

trolled by a threshold parameter (*gap*). When the queue modeling phase ends, the algorithm finds the next empty state of the queue to restart the modeling process.

- **Step 4** ***(bandwidth estimation).*** Once the busy periods are captured, the algorithm performs bandwidth estimation by computing the fraction of the amount of downlink traffic observed between the first and last test packets of the busy period and the elapsed time. We note that, depending on the length of the traffic sample, our method may identify several busy periods. The final result will be the mean of the estimated values.

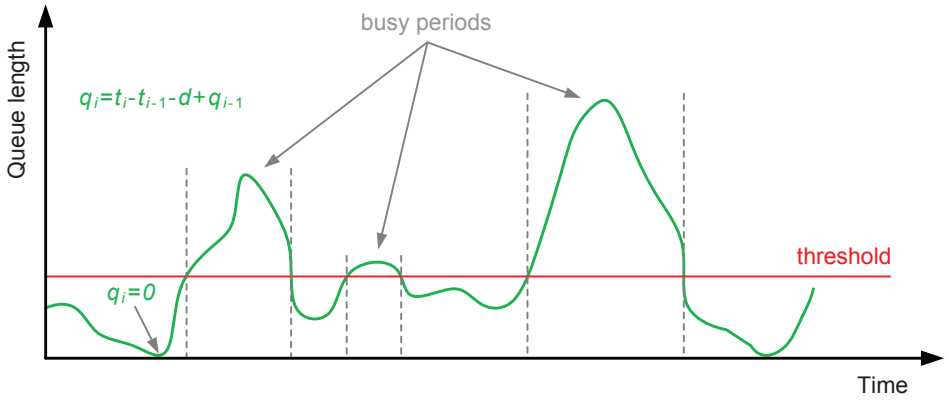
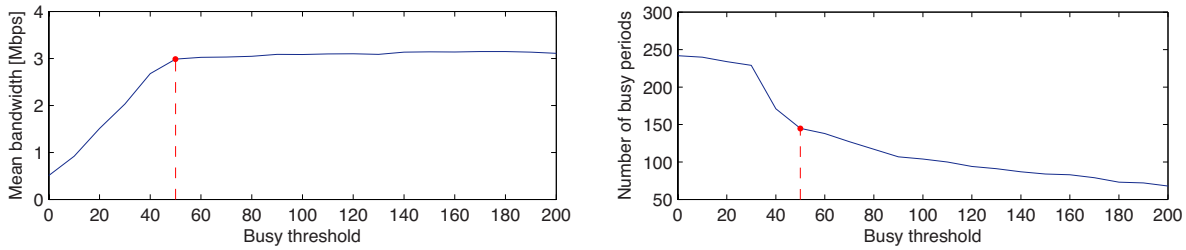


Figure 6.4. Busy period detection by modeling the queue dynamics

6.3 Evaluation Results

To evaluate the operation of our heuristic bandwidth estimation scheme, we conducted several measurement scenarios in a controlled environment. In this section, we show an example from our test results obtained on a real traffic trace captured in a 3G mobile network. As discussed above, the algorithm was designed to estimate the currently available bandwidth by exploiting the user-generated downlink traffic. To examine typical user behaviors, we used a multi-functional network traffic emulator presented in [105]. This tool can accurately simulate different types of user activity such as web browsing, or the use of video streaming services (e.g. YouTube) and social networking applications (e.g. Facebook).

The measurements were performed on a smartphone with HSDPA support and Android operating system where we generated realistic web traffic based on user behavior emulation. The test packets were sent periodically from the server towards the mobile



(a) Mean estimated available bandwidth (b) Number of busy period samples

Figure 6.5. The choice of busy threshold

device over UDP with an inter-arrival time of 100 ms. We measured the IAT distribution at the receiver with no cross-traffic and observed that the time spaces between consecutive UDP packets were only slightly changed (in the order of few milliseconds). However, to take this effect into account, mostly caused by the variation of signal quality, a positive busy threshold was applied in the queue modeling phase according to Algorithm 1. In order to mitigate the network load induced by the test traffic, small-sized (i.e. 60 bytes) UDP packets were injected into the user's downlink stream. At the mobile device we captured 60 minutes long packet traces for evaluation purposes and carried out an extensive analysis by investigating many different aspects.

The following results present the traffic intensity for the measured and estimated time series, the busy period statistics, as well as the histograms and distribution functions of the download rate. To identify the busy periods of the bottleneck queue, we used a positive threshold of 50, and for outlier detection we defined the maximum acceptable inter-arrival time as 1000 ms. During our evaluation tests, we experimented with different busy thresholds and concluded that a positive value has to be applied in order to filter out the impact of some undesirable phenomena like jitter. However, in general, above a certain threshold we get very similar estimation results including the distribution and mean of the estimated bandwidth values (Figure 6.5a), but a higher value leads to a smaller number of detected busy periods over a given time interval (Figure 6.5b). To obtain the best outcome, it is practical to choose the lowest possible threshold which otherwise can be considered as sufficient to avoid the issues mentioned above.

Figure 6.6 shows the measured downlink traffic intensity in one second resolution and the estimated available bandwidth calculated for the busy periods. Web traffic is eligible to demonstrate the capabilities of our bandwidth estimation method since typical users frequently check their emails and favorite social networking sites, or simply browse the

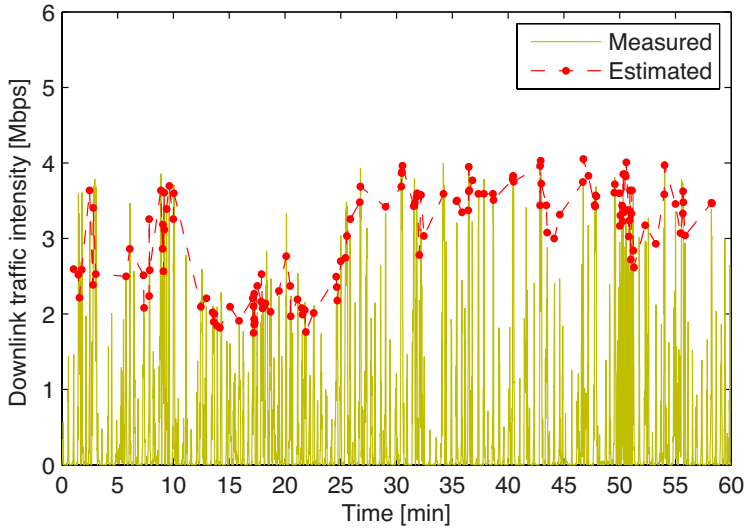


Figure 6.6. Traffic intensity

web. Our main goal was to design such an algorithm, which can give an estimate for the unused bandwidth even if the user generates only a small amount of network traffic, for example, by web browsing. The figure indicates that the downlink traffic highly fluctuates due to the characteristics of user activity, but we can identify many intervals when a page load utilizes the instantaneous available bandwidth. This means that during several periods of time the bottleneck queue is busy, or in other words, it works at the maximum service rate. The figure depicts the estimated bandwidth calculated for these busy periods. We emphasize that it is really hard to give an accurate estimation, because in a mobile network available bandwidth is continuously changing and affected by many conditions like motion speed, the number of users in the current cell, signal strength, handovers, and so on [101, 102]. In spite of this fact, one can see that the busy periods identified by our heuristic algorithm covers well the highest download rates offered by the network.

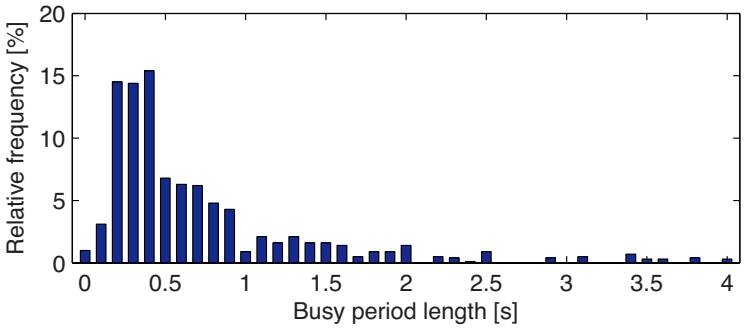
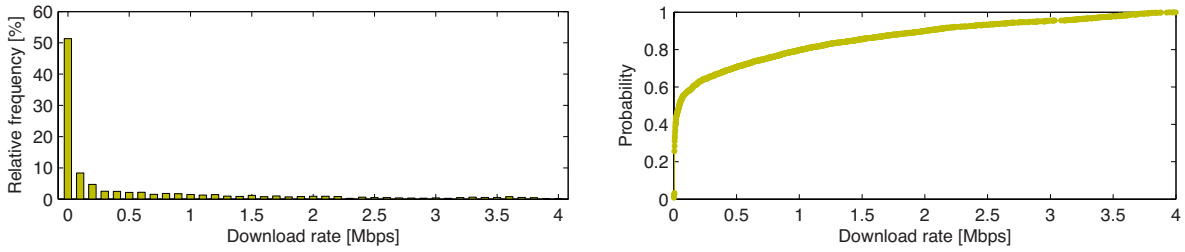
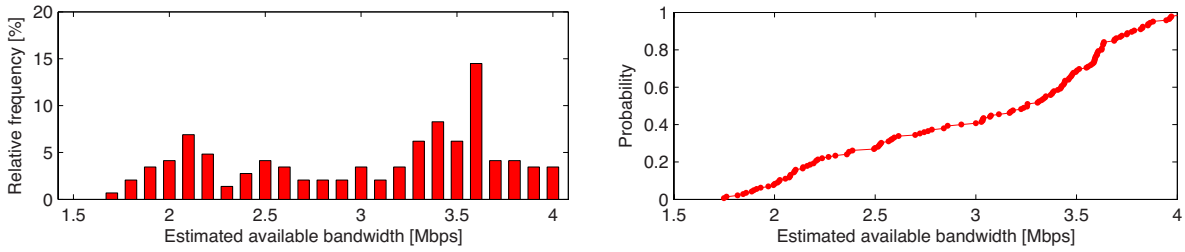


Figure 6.7. Busy period statistics



(a) Histogram (b) CDF

Figure 6.8. Measured rate characteristics



(a) Histogram (b) CDF

Figure 6.9. Estimated bandwidth characteristics

Figure 6.7 presents the relative frequencies of busy period lengths. The results clearly show that busy periods are quite short in the case of web traffic. Specifically, almost 50% and 75% of all captured busy periods are shorter than half and one second, respectively. Web browsing typically results in bursty traffic since users spend at least a few seconds on a page before proceeding. Nevertheless, the length of the downloading periods is still sufficient to calculate proper bandwidth estimates.

Figure 6.8 and Figure 6.9 depict the histogram and the CDF of the measured download rate and the estimated available bandwidth, respectively. Looking at Figure 6.8, we can find low download rates much more frequent compared to high rates in the measured packet trace. This is due to the phenomenon discussed earlier in the chapter, namely, the maximum bandwidth is utilized only in the cases of traffic bursts, which are separated by idle periods. For example, more than 70% of transmission rates fall below 0.5 Mbps, because once a page is loaded no further network traffic is usually generated or only small amount of data is need to be exchanged (e.g. for online advertisements). Our purpose was to capture those intervals when downloading consumes the available bandwidth. We ran 100 rounds of the widely used *Speedtest* [106] on the mobile device with half minute breaks before and after the one hour long measurement period. The perceived available

downlink bandwidth was between 1.6 and 4.2 Mbps in accordance with our estimation results calculated for the busy periods, see Figure 6.9a. Furthermore, the mean bandwidth provided by Speedtest was 3.1 Mbps, which is also very close to our estimate of 3 Mbps (Figure 6.5a). As pointed out in the discussion of Figure 6.7, busy periods are short in time. Moreover, Figure 6.9b suggests that web traffic originated from a typical smartphone user contains small number of busy periods, hence it is crucial how accurately the detection method can capture them. While each estimated bandwidth value exceeds 1.7 Mbps, about 85% of measured rates are below this limit (Figure 6.8b), accordingly, do not fall into any of the identified busy periods.

6.4 Conclusion

Bandwidth estimation in cellular networks is challenging due to the nature of radio communication. Currently available bandwidth is a continuously changing metric affected by numerous environmental factors. In this chapter, we proposed a heuristic approach, which can exploit the user-generated traffic and is capable of modeling the dynamics of the bottleneck queue and capturing its busy periods. We demonstrated the operability of our algorithm on a packet trace gathered in a 3G mobile network by using a realistic traffic emulator. It has been found that busy periods can be relatively short, but the presented method is able to capture them with high reliability. The results suggest that proper identification of busy periods makes it possible to estimate the available bandwidth.

Chapter 7

Summary

7.1 Main Contributions and Conclusions

Reliable end-to-end communication over the Internet was ensured by the congestion control algorithms of TCP throughout the past decades. However, due to the rapid growth of computer networks, several different TCP versions have been worked out to enhance the performance under specific conditions. This long development process has been resulted only in a barely manageable set of transport protocols instead of an optimal and universal solution. As the researchers had recognized that TCP cannot keep up with the evolving network technologies, they started to look for new research directions and advocated different approaches.

In this dissertation, we have investigated a novel data transfer paradigm relying on digital fountains for future networks, which omits congestion control from the transport layer together with its inherent drawbacks. We have presented a network architecture based on our suggested transmission mechanism and identified the necessary core components. We also have designed a transport protocol called DFCP for this new architecture with all of its features like signaling, coding, data transfer and flow control. A detailed discussion was given about the operating principles and the potential benefits. In order to explore the main properties of DFCP, a highly reliable multi-platform evaluation environment has been built. By using this framework, we have carried out a comprehensive performance analysis to compare DFCP and TCP in terms of goodput performance, buffer demand, flow transfer efficiency, fairness, network utilization and scalability. Since today's networks are highly variable, we have characterized the two transfer paradigms under dynamic traffic conditions as well, focusing on the features of stability, convergence, responsiveness and saturation time. Our results clearly revealed that the digital fountain

based approach is a promising alternative to congestion control. The major advantages lie in the moderate packet loss and delay sensitivity, the good performance even in networks with tiny buffers, the low flow completion times, the fair bandwidth sharing, the stability and the fast convergence.

Furthermore, we have worked out a bandwidth estimation method for mobile networks, which can estimate the unused bandwidth by exploiting the user-generated downlink network traffic with negligible extra load. The operation of the algorithm is investigated on real traffic traces captured by a mobile device in a 3G network. Our key finding is that, even though the busy periods are typically very short, proper identification can make it possible to estimate the available bandwidth with reasonable accuracy.

7.2 Possible Applications

The envisioned architecture built upon the transport mechanism of DFCP is a good candidate for data communication of future networks due to its capability of supporting novel applications and use-cases. In this section, we discuss these potential application areas.

Multipath transport has received significant attention in recent years. As a result of these activities, MPTCP has been standardized by IETF [30]. Moreover, now it is available as a kernel implementation to Linux [107] giving the chance of proliferation. By *multipath communication*, network resiliency, efficient transfer or load balancing can be provided. However, the congestion control scheme of MPTCP is currently based on TCP Reno, which is the root cause of some severe issues of the protocol (e.g. poor performance in high BDP networks). As we see, many of the congestion control related problems of MPTCP can be mitigated by combining it with our digital fountain based transfer mechanism.

In *data centers*, the communication between network nodes can be significant. This type of operation is supported by well-designed network topologies. However, it is not enough to make efficient transfer possible between inner nodes. DCTCP (Data Center TCP) is a recent approach intended to fulfill the specific requirements of data center networks, which can maintain small queue length and low latency for short flows [108]. The key points where DFCP would improve the performance of DCTCP includes the possibility of further reducing the buffer size, a moderate queue oscillation, a lower flow completion time and the fact that the queue length is independent of the number of concurrent flows.

Another and potential application area of DFCP is *wireless networks*. The performance of TCP is very poor in wireless environment, which is due to the basic inherent design principle of TCP assuming that packet loss is a result of network congestion [16]. In contrast, in wireless communication we find significant packet loss caused by not congestion but erroneous wireless channels resulting in high bit error rate that may arrive in bursts. Wireless links often use data link level solutions to tackle this problem like layer 2 methods with forward error correction and automatic repeat request (ARQ) [109]. However, such solutions hardly cooperate with TCP. For example, these mechanisms add an extra delay to TCP's RTT estimate assuming a far higher latency on the path than the case is. Moreover, TCP easily triggers a retransmission at the same time when ARQ is already retransmitting the same data. In this case, TCP will experience an ACK timeout and it is forced to recommence from the slow-start mode and from the point of packet loss. In general, TCP is very sensitive to packet loss and has a very poor performance even if the packet loss rate does not exceed 1 percent. In contrast, DFCP is insensitive to packet loss in a wide range of packet loss rates as it was demonstrated in Chapter 4. This property gives a great motivation for applying DFCP as the transport protocol in wireless environments and it also implies that the application of DFCP eliminates the need for all additional and essential mechanisms (e.g. ARQ methods in layer 2) with their interoperability problems, which are unavoidable if TCP is used.

The new transport mechanism also has a high potential to deploy it in *optical networks*. This is due to the attractive feature of our concept that makes it possible to build a bufferless network architecture. Since DFCP inherently counts for congestion due to packet loss in the network, there is no need to apply buffers in network nodes to avoid packet loss. Therefore, the great challenge currently preventing the deployment of building all-optical cross-connects in optical networks can be solved [86]. This feature also makes a possibility to build a more cheaper wired Internet, because it is unnecessary to use expensive and power-hungry line card memories in network routers as we do it in our TCP-based Internet. Buffers can be short or even totally eliminated in this networking paradigm. Moreover, as a consequence of this approach the extra queuing delay in router buffers, which is a significant, hardly treatable and highly variable performance determining factor in our current Internet, can be avoided resulting in an easier network design and dimensioning process.

Considering the *deployment options* of DFCP, as we pointed out in Section 3.5.3, SDN is a very attractive environment. In general, inter-protocol fairness between different

TCP versions is an important issue, but DFCP and TCP cannot work together within the same network due to the fundamental difference in the applied paradigms. It means that DFCP would grab all capacity from TCP since it operates in the overloaded regime (see Chapter 4). One possible solution to avoid such incompatibility is to deploy DFCP alone in a given target environment like SDN. Although we do not believe that our protocol should certainly be used in the whole Internet, its co-existence with TCP could be realized by building overlay networks on top of the current physical infrastructure. More precisely, during a transition period physical resources such as link capacities and router memories can be split between the traditional TCP-based and the proposed DFCP-based architectures.

7.3 Open Issues and Future Directions

Our research, presented in this dissertation, pointed out that digital fountain based transport is a promising alternative to TCP, which merits further investigation. However, beyond the several potential benefits it also brings a lot of challenges and raises many questions.

One of the most important unsolved problems is the consequence of the maximal rate sending principle of DFCP since it is easy to construct a network topology where this approach results in an undesirable bandwidth waste also known as *dead packet phenomenon*. In fact, it is due to the absence of congestion control, however, there are several possible ways to tackle this issue as outlined in Section 3.2. Emerging paradigms and technologies such as SDN may provide an excellent framework to effectively control the transmission rates of individual flows. To this end, a suitable algorithm needs to be worked out and investigated by careful experiments.

Regarding the data transfer mechanism, it would be interesting and practical to extend the features with the capability of *adaptive parameter optimization* during the communication so as to make efficient operation possible under dynamically changing conditions. The coding scheme of DFCP currently implements standard Raptor codes [47], which is eligible to study the main principles of digital fountain based transport. However, it would be useful to experiment with the most advanced version called RaptorQ [50], and to take into account the *future evolution of Raptor codes* from the point of view of protocol design as seeking for more and more efficient decoding algorithms is a very active research area [110, 111].

We note that the measurements were performed on simple network topologies. Although most researchers evaluate their proposals by using these widely known reference topologies, the next step could be a *large-scale analysis* on a platform like PlanetLab [112]. Another possible direction is to deeply investigate how the new transfer mechanism fits into *specific environments* such as data centers or all-optical networks.

Bibliography

[1] J. Postel, "Transmission Control Protocol", *RFC 793, IETF*, 1981.

[2] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, "Host-to-Host Congestion Control for TCP", *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[3] B. Raghavan, A. C. Snoeren, "Decongestion Control", *Proceedings of the 5th ACM Workshop on Hot Topics in Networks*, pp. 61–66, Irvine, CA, USA, 2006.

[4] D. Clark, S. Shenker, A. Falk, "GENI Research Plan (Version 4.5)", April 23, 2007.

[5] T. Bonald, M. Feuillet, A. Proutiere, "Is the 'Law of the Jungle' Sustainable for the Internet?", *Proceedings of the 28th IEEE Conference on Computer Communications*, pp. 28–36, Rio de Janeiro, Brazil, 2009.

[6] Network Simulation Cradle, `http://www.wand.net.nz/~stj2/nsc/`

[7] ns-2 Network Simulator, `http://www.isi.edu/nsnam/ns/`

[8] Emulab Network Emulation Testbed, `http://www.emulab.net/`

[9] V. Jacobson, "Congestion Avoidance and Control", *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, 1988.

[10] S. Molnár, B. Sonkoly, T. A. Trinh, "A Comprehensive TCP Fairness Analysis in High Speed Networks", *Elsevier Computer Communications*, vol. 32, no. 13–14, pp. 1460–1484, 2009.

[11] Y.-T. Li, D. Leith, R. N. Shorten, "Experimental Evaluation of TCP Protocols for High-Speed Networks", *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1109–1122, 2007.

[12] T. Kelly, "Scalable TCP: Improving Performance in High-Speed Wide Area Networks", *ACM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[13] S. Floyd, "HighSpeed TCP for Large Congestion Windows", *RFC 3649, IETF*, 2003.

[14] C. Jin, D. X. Wei, S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance", *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[15] I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks*, pp. 1–6, Lyon, France, 2005.

[16] Y. Tian, K. Xu, N. Ansari, "TCP in Wireless Environments: Problems and Solutions", *IEEE Communications Magazine*, vol. 43, no. 3, pp. 27–32, 2005.

[17] B. Francis, V. Narasimhan, A. Nayak, I. Stojmenovic, "Techniques for Enhancing TCP Performance in Wireless Networks", *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems Workshop*, pp. 222–230, Macau, China, 2012.

[18] T. Goff, J. Moronski, D. Phatak, "Freeze-TCP: A True End-to-End Enhancement Mechanism for Mobile Environments", *Proceedings of the 19th IEEE International Conference on Computer Communications*, pp. 1537–1545, Tel-Aviv, Israel, 2000.

[19] I. F. Akyildiz, G. Morabito, S. Palazzo, "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks", *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 307–321, 2001.

[20] J. Liu, S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks", *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, 2001.

[21] A. Venkataramani, R. Kokku, M. Dahlin, "TCP Nice: A Mechanism for Background Transfers", *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 329–344, 2002.

[22] A. Kuzmanovic, E. W. Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer", *Proceedings of the 22th IEEE International Conference on Computer Communications*, pp. 1691–1701, San Francisco, CA, USA, 2003.

[23] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", *RFC 6817, IETF*, 2012.

[24] D.-M. Chiu, R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", *Journal of Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.

[25] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[26] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options", *RFC 2018, IETF*, 1996.

[27] S. Floyd, T. Henderson, A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", *RFC 3782, IETF*, 2004.

[28] L. Xu, K. Harfoush, I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", *Proceedings of the 23rd IEEE International Conference on Computer Communications*, vol. 4, pp. 2514–2524, Hong Kong, China, 2004.

[29] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, R. Wang, "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks", *ACM Journal of Wireless Networks*, vol. 8, no. 5, pp. 467–479, 2002.

[30] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, "Architectural Guidelines for Multipath TCP Development", *RFC 6182, IETF*, 2011.

[31] L. S. Brakmo, L. L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[32] K. Tan, J. Song, Q. Zhang, M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks", *Technical Report, Microsoft Research*, pp. 1–12, 2005.

[33] S. Liu, T. Basar, R. Srikant, "TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks", *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools*, pp. 1–13, Pisa, Italy, 2006.

[34] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, "Stream Control Transmission Protocol", *RFC 2960, IETF*, 2000.

[35] D. Katabi, M. Handley, C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks", *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 89–102, 2002.

[36] S. Molnár, Z. Móczár, "Three-dimensional Characterization of Internet Flows", *Proceedings of the 46th IEEE International Conference on Communications*, pp. 1–6, Kyoto, Japan, 2011.

[37] N. Dukkipati, N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control", *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[38] J. Iyengar, I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", *Internet Draft, IETF*, 2015.

[39] G. Carlucci, L. D. Cicco, S. Mascolo, "HTTP over UDP: an Experimental Investigation of QUIC", *Proceedings of the 30th ACM/SIGAPP Symposium on Applied Computing*, pp. 1–6, Salamanca, Spain, 2015.

[40] M. Wanga, J. Wanga, S. Han, "Adaptive Congestion Control Framework and a Simple Implementation on High Bandwidth-Delay Product Networks", *Elsevier Computer Networks*, vol. 64, pp. 308–321, 2014.

[41] A. Sivaraman, K. Winstein, P. Thaker, H. Balakrishnan, "An Experimental Study of the Learnability of Congestion Control", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 479–490, 2014.

[42] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, M. Schapira, "PCC: Re-architecting Congestion Control for Consistent High Performance", *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, pp. 395–408, Oakland, CA, USA, 2015.

[43] D. J. C. MacKay, "Fountain Codes", *IEE Proceedings – Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[44] M. Luby, "Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs", *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 171–175, Barcelona, Spain, 1998.

[45] M. Luby, "LT Codes", *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pp. 271–280, Vancouver, BC, Canada, 2002.

[46] A. Shokrollahi, "Raptor Codes", *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[47] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery", *RFC 5053, IETF*, 2007.

[48] C. Bouras, N. Kanakis, V. Kokkinos, A. Papazois, "Embracing RaptorQ FEC in 3GPP Multicast Services", *Wireless Networks, The Journal of Mobile Communication, Computation and Information*, vol. 19, no. 5, pp. 1023–1035, 2013.

[49] L. R. Wilhelmsson, "Evaluating the Performance of Raptor Codes for DVB-H by Using the Gilbert-Elliott Channel", *Proceedings of the 66th IEEE Vehicular Technology Conference*, pp. 1932–1936, Baltimore, MD, USA, 2007.

[50] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery", *RFC 6330, IETF*, 2011.

[51] L. López, A. Fernández, V. Cholvi, "A Game Theoretic Comparison of TCP and Digital Fountain Based Protocols", *Elsevier Computer Networks*, vol. 51, no. 12, pp. 3413–3426, 2007.

[52] D. Kumar, T. Chahed, E. Altman, "Analysis of a Fountain Codes Based Transport in an 802.11 WLAN Cell", *Proceedings of the 21st International Teletraffic Congress*, pp. 1–8, Paris, France, 2009.

[53] A. Botos, Z. A. Polgar, V. Bota, "Analysis of a Transport Protocol Based on Rateless Erasure Correcting Codes", *Proceedings of the 2010 IEEE International Conference on Intelligent Computer Communication and Processing*, vol. 1, pp. 465–471, Cluj-Napoca, Romania, 2010.

[54] Y. Cui, X. Wang, H. Wang, G. Pan, Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol", *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems*, pp. 366–375, Macau, China, 2012.

[55] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, J. Barros, "Network Coding Meets TCP: Theory and Implementation", *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.

[56] M. Shreedhar, G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin", *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.

[57] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, "On the Scalability of Fair Queuing", *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks*, pp. 1–6, San Diego, CA, USA, 2004.

[58] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, "Evaluating the Number of Active Flows in a Scheduler Realizing Fair Statistical Bandwidth Sharing", *Proceedings of the ACM SIGMETRICS 2005 International Conference on Measurement and Modeling of Computer Systems*, pp. 217–228, Banff, AB, Canada, 2005.

[59] J. McCullough, B. Raghavan, A. C. Snoeren, "The Role of End-to-End Congestion Control in Networks with Fairness-Enforcing Routers", *Technical Report, University of California, San Diego*, pp. 1–14, 2013.

[60] Y. Gu, X. Hong, R. L. Grossman, "Experiences in Design and Implementation of a High Performance Transport Protocol", *Proceedings of the 2004 ACM/IEEE Conference on High Performance Computing, Networking and Storage*, Pittsburgh, PA, USA, 2004.

[61] K. Ramakrishnan, S. Floyd, D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", *RFC 3168, IETF*, 2001.

[62] B. Briscoe, A. Jacquet, T. Moncaster, A. Smith, "Re-ECN: Adding Accountability for Causing Congestion to TCP/IP", *Internet Draft, IETF*, 2014.

[63] B. Briscoe, R. Woundy, A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases", *RFC 6789, IETF*, 2012.

[64] M. Ghobadi, S. H. Yeganeh, Y. Ganjali, "Rethinking End-to-End Congestion Control in Software-Defined Networks", *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 61–66, Redmond, WA, USA, 2012.

[65] A. Shokrollahi, "LDPC Codes: An Introduction", *Technical Report, Digital Fountain Inc.*, pp. 1–34, 2003.

[66] K. Pawlikowski, H.-D. Joshua Jeong, J.-S. Ruth Lee, "On Credibility of Simulation Studies of Telecommunication Networks", *IEEE Communications Magazine*, vol. 40, no. 1, pp. 132–139, 2002.

[67] M. Bateman, S. Bhatti, "TCP Testing: How Well Does ns2 Match Reality?", *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 276–284, Perth, Australia, 2010.

[68] S. Floyd, E. Kohler, "Tools for the Evaluation of Simulation and Testbed Scenarios", *Technical Report, IETF*, 2006.

[69] M. P. Fernandez, S. Wahle, T. Magedanz, "A New Approach to NGN Evaluation Integrating Simulation and Testbed Methodology", *Proceedings of the 11th International Conference on Networks*, pp. 22–27, Saint-Gilles, Réunion Island, France, 2012.

[70] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms", *RFC 5166, IETF*, 2008.

[71] J.-Y. L. Boudec, "Rate Adaptation, Congestion Control and Fairness: A Tutorial", *Technical Report, École Polytechnique Fédérale de Lausanne (EPFL)*, pp. 1–44, 2015.

[72] D. X. Wei, P. Cao, S. H. Low, "Time for a TCP Benchmark Suite?", *Technical Report, California Institute of Technology*, pp. 1–6, 2005.

[73] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks", *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1396, 1995.

[74] P. McKenney, "Stochastic Fairness Queueing", *Internetworking: Research and Experience*, vol. 2, pp. 113–131, 1991.

[75] R. Adams, "Active Queue Management: A Survey", *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013.

[76] Dummynet Network Emulator, `http://info.iet.unipi.it/~luigi/dummynet/`

[77] M. Lacage, "Experimentation Tools for Networking Research", *Ph.D. Thesis* (available at `http://cutebugs.net/files/thesis.pdf`), 2010.

[78] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[79] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost", *Passive and Active Measurement Conference, Lecture Notes in Computer Science*, vol. 7799, pp. 31–41, 2013.

[80] S. Chowdhury, M. Bari, R. Ahmed, R. Boutaba, "PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks", *Proceedings of the 14th Network Operations and Management Symposium*, pp. 1–9, Krakow, Poland, 2014.

[81] N. L. M. van Adrichem, C. Doerr, F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks", *Proceedings of the 14th Network Operations and Management Symposium*, pp. 1–8, Krakow, Poland, 2014.

[82] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[83] C.-Y. Hong, M. Caesar, P. B. Godfrey, "Software Defined Transport: Flexible and Deployable Flow Rate Control", *Proceedings of the Open Networking Summit 2014*, pp. 1–2, Santa Clara, CA, USA, 2014.

[84] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, R. Steinmetz, "Modelling the Internet Delay Space Based on Geographical Locations", *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 301–310, Weimar, Germany, 2009.

[85] T. V. Lakshman, U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss", *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, 1997.

[86] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing Router Buffers", *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[87] H. Park, E. F. Burmeister, S. Bjorlin, J. E. Bowers, "40-Gb/s Optical Buffer Design and Simulation", *Proceedings of the 4th International Conference on Numerical Simulation of Optoelectronic Devices*, pp. 19–20, Santa Barbara, CA, USA, 2004.

[88] HTTP Archive, `http://www.httparchive.org/interesting.php`

[89] G. Carofiglio, L. Muscariello, "On the Impact of TCP and Per-Flow Scheduling on Internet Performance", *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 620–633, 2012.

[90] Y. Huang, R. Guérin, "Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger?", *Proceedings of the 13th IEEE International Conference on Network Protocols*, pp. 225–235, Boston, MA, USA, 2005.

[91] J. Gettys, K. Nichols, "Bufferbloat: Dark Buffers in the Internet", *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, 2012.

[92] P. Loskot, M. A. M. Hassanien, F. Farjady, M. Ruffini, D. Payne, "Long-Term Drivers of Broadband Traffic in Next-Generation Networks", *Annals of Telecommunications*, vol. 70, no. 1, pp. 1–10, 2015.

[93] D. Bansal, H. Balakrishnan, S. Floyd, S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms", *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 263–274, 2001.

[94] Y. R. Yang, M. S. Kim, S. S. Lam, "Transient Behaviors of TCP-friendly Congestion Control Protocols", *Proceedings of the 20th IEEE International Conference on Computer Communications*, vol. 3, pp. 1716–1725, Anchorage, AK, USA, 2001.

[95] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, "Bandwidth Estimation: Metrics, Measurement, and Tools", *IEEE Network*, vol. 17, pp. 27–35, 2003.

[96] C. D. Guerrero, M. A. Labrador, "On the Applicability of Available Bandwidth Estimation Techniques and Tools", *Elsevier Computer Communications*, vol. 33, no. 1, pp. 11–22, 2010.

[97] J. Strauss, D. Katabi, F. Kaashoek, "A Measurement Study of Available Bandwidth Estimation Tools", *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pp. 39–44, Karlsruhe, Germany, 2003.

[98] M. Mathis, M. Allman, "A Framework for Defining Empirical Bulk Transfer Capacity Metrics", *RFC 3148, IETF*, 2001.

[99] M. Allman, "Measuring End-to-End Bulk Transfer Capacity", *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 139–143, San Francisco, CA, USA, 2001.

[100] R. Gardner, F. Garcia, "Bulk Transfer Capacity Estimation in IPv6 Networks", *Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, pp. 1–6, Bucharest, Romania, 2006.

[101] U. Hentschel, A. Schmidt, A. Polze, "Predictable Communication for Mobile Systems", *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pp. 24–28, Newport Beach, CA, USA, 2011.

[102] J. A. Negreira, J. Pereira, S. Pérez, P. Belzarena, "End-to-End Measurements Over GPRS-EDGE Networks", *Proceedings of the 4th IFIP/ACM International Latin American Conference on Networking*, pp. 121–131, San José, Costa Rica, 2007.

[103] A. Gember, A. Akella, J. Pang, A. Varshavsky, R. Caceres, "Obtaining In-Context Measurements of Cellular Network Performance", *Proceedings of the 12th ACM International Conference on Internet Measurement*, pp. 287–300, Boston, MA, USA, 2012.

[104] E. Bergfeldt, S. Ekelin, J. M. Karlsson, "A Performance Study of Bandwidth Measurement Tools over Mobile Connections", *Proceedings of the 69th IEEE International Vehicular Technology Conference*, pp. 1–5, Barcelona, Spain, 2009.

[105] S. Molnár, P. Megyesi, G. Szabó, "Multi-Functional Emulator for Traffic Analysis", *Proceedings of the 48th IEEE International Conference on Communications*, pp. 2397–2402, Budapest, Hungary, 2013.

[106] Ookla Speedtest, `http://www.speedtest.net/`

[107] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, pp. 1–14, San Jose, CA, USA, 2012.

[108] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, "Data Center TCP (DCTCP)", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 63–74, 2010.

[109] C. Barakat, E. Altman, "Bandwidth Tradeoff Between TCP and Link-Level FEC", *Elsevier Computer Networks*, vol. 39, no. 2, pp. 133–150, 2002.

[110] X. Guo, G.-X. Zhang, C. Tian, L. Zhang, W.-D. Zhao, "Fast Decoding for RaptorQ Codes Using Matrix Dimensionality Reduction", *IET Electronics Letters*, vol. 50, no. 16, pp. 1139–1141, 2014.

[111] Y. Xing, N. Ge, "An On-Line Decoding Algorithm for 3GPP MBMS Raptor Codes", *Proceedings of the 81st IEEE Vehicular Technology Conference*, pp. 1–5, Glasgow, Scotland, 2015.

[112] PlanetLab: An Open Platform for Developing, Deploying and Accessing Planetary-Scale Services, `https://www.planet-lab.org/`

Publications

[P1] S. Molnár, **Z. Móczár**, B. Sonkoly, "Living with Congestion: Digital Fountain based Communication Protocol", accepted to *Elsevier Computer Communications*, 2016.

[P2] **Z. Móczár**, S. Molnár, "On the Dynamic Behavior of Digital Fountain Based Communication", *Proceedings of the 58th IEEE Global Communications Conference, Exhibition and Industry Forum (GLOBECOM 2015)*, pp. 1–6, San Diego, CA, USA, 2015.

[P3] **Z. Móczár**, S. Molnár, "Bandwidth Estimation in Mobile Networks by Busy Period Detection", *Proceedings of the 25th IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2014)*, pp. 1354–1358, Washington D.C., USA, 2014.

[P4] **Z. Móczár**, S. Molnár, "Towards the Transport Protocols of Future Internet", *Infocommunications Journal*, vol. 6, no. 3, pp. 3–9, 2014.

[P5] **Z. Móczár**, S. Molnár, B. Sonkoly, "Multi-Platform Performance Evaluation of Digital Fountain Based Transport", *Proceedings of the Science and Information Conference 2014 (SAI 2014)*, pp. 690–697, London, UK, 2014.

[P6] S. Molnár, **Z. Móczár**, B. Sonkoly, "How to Transfer Flows Efficiently via the Internet?", *Proceedings of the 3rd IEEE International Conference on Computing, Networking and Communications (ICNC 2014)*, pp. 462–466, Honolulu, HI, USA, 2014.

[P7] S. Molnár, **Z. Móczár**, A. Temesváry, B. Sonkoly, Sz. Solymos, T. Csicsics, "Data Transfer Paradigms for Future Networks: Fountain Coding or Congestion Control?", *Proceedings of the 12th IFIP International Conference on Networking (Networking 2013)*, pp. 1–9, New York, NY, USA, 2013.