Computer Communications 80 (2016) 82–100

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Living with congestion: Digital Fountain based Communication Protocol

Sándor Molnár, Zoltán Móczár*, Balázs Sonkoly

High Speed Networks Laboratory, Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Budapest, Hungary

ARTICLE INFO

Article history:
Received 5 August 2014
Revised 13 January 2016
Accepted 15 January 2016
Available online 2 February 2016

Keywords:
Fountain coding
Congestion control
Transport protocol
Network architecture
Performance evaluation

ABSTRACT

In this paper, we propose a networking paradigm built upon a fountain code based data transfer mechanism. We advocate that, instead of controlling the congestion as it is applied in recent Internet by the Transmission Control Protocol (TCP), we can utilize congestion and neglect control algorithms with all of their drawbacks. We present our envisioned network architecture relying on a novel transport protocol called *Digital Fountain based Communication Protocol (DFCP)* and highlight some potential application areas. We have implemented DFCP in the Linux kernel and we provide its validation results gained from three different testing platforms including our laboratory testbed, the Emulab network emulation environment and the ns-2 network simulator. Moreover, we present and discuss a comprehensive performance evaluation of DFCP in comparison with widely used TCP versions.

1. Introduction

A fearful phenomenon called *congestion* has been observed from the beginning of computer networks when an aggregate demand for a resource exceeds its available capacity. Congestion can result in long delays, lost packets and consequently degraded quality of the service and wasted resources of the network. The phenomenon became even more dreadful in the early days of the Internet by causing *congestion collapse* [1] when the increase in network load led to the decrease of useful work in the entire network. It was the reason to introduce *congestion control*, which is a set of mechanisms to limit the demand-capacity mismatch by controlling traffic sources when such a mismatch occurs.

In the history of the Internet, closed-loop congestion control was the successful paradigm to avoid congestion collapse and the related performance degradation due to the overload of network resources. Congestion control is mostly performed by the *Transmission Control Protocol (TCP)*, which transports more than 80% of Internet traffic. The basic idea behind the congestion control mechanism of TCP is that the sender gradually increases the transmission rate until a packet loss is detected, and when it happens, the sending rate is cut in half starting the cycle again. Packets can be lost due to both congestion and environmental factors (e.g. lossy wireless links), which is typically indicated by an explicit feedback signal, or inferred by the sender from the reception of three duplicate acknowledgments. In this paper, we focus on this traditional loss-based approach, however, we note that some TCP variants adjust their transmission rate by measuring round-trip delay variations, or use hybrid solutions. The success of TCP was not even questioned until the fast development of networks, mobile devices and user applications resulted in heterogeneous and complex environments over the last decades. In order to fit these changes, significant research was carried out to further develop TCP, and therefore, several different TCP versions have been proposed [2–5]. However, it turned out that it will be very difficult to modify TCP to work efficiently as a universal transport protocol.

In this paper, we propose a different data transfer paradigm, namely, *instead of avoiding congestion we advocate to allow it*. The main idea is that it might be possible to perform efficient communication even in the presence of congestion. First, we enable *sources to transmit their data at the maximum possible rate*, which will shift the operation to the *overloaded regime*. Theoretically, such a solution can be considered as the most efficient one, because the network would always be fully utilized by hosts sending at maximal rates, and therefore, each additional capacity would immediately be consumed. In practice, however, this greedy behavior can lead to bandwidth waste that needs to be addressed by a proper rate limitation mechanism (see Sections 3.2 and 7.6 for the discussion). Moreover, we suggest to apply *efficient digital fountain based coding schemes* for encoding the application-level data. The proposed approach implies that packet loss becomes inconsequential, which can considerably simplify network routers

* Corresponding author. Tel.: +36 205215496.
E-mail addresses: molnar@tmit.bme.hu (S. Molnár), moczar@tmit.bme.hu (Z. Móczár), sonkoly@tmit.bme.hu (B. Sonkoly).

http://dx.doi.org/10.1016/j.comcom.2016.01.008
0140-3664/

and can result in highly reduced buffer sizes closely aligned to the concept of all-optical networking. Concerning stability, we can also expect improvements since transmitting at the maximum possible rate would result in more predictable traffic patterns by avoiding the high extent of rate variation often seen in TCP transmissions. This property would make network design and dimensioning easier tasks as well. Moreover, we suggest to use *fair schedulers* in network nodes to ensure fairness among competing flows. Since simple scalable schedulers are already available in routers, the seamless deployment of the concept can be feasible.

In order to take research on alternative data transfer paradigms forward, we studied the possibility of applying a *fountain code* [6] based transport protocol instead of the congestion control based TCP. To the best of our knowledge, such a transport protocol has not been developed and implemented with a comprehensive performance evaluation study yet, thus we made our prototype implementation and carried out a thorough analysis. First, we introduced the *Digital Fountain based Communication Protocol (DFCP)* in [7] with some preliminary analytical, simulation and testbed results. In this paper, the whole idea of the new architecture with a detailed description and performance evaluation of DFCP is presented. Our intention is to deliver the message that the proposed transport mechanism can be a potential alternative to TCP for future networks in several application areas.

In summary, the main contributions of this paper are

- the introduction of our envisioned network architecture and data transfer paradigm with discussions on possible future applications and challenges,
- a detailed description of our proposed transport protocol (DFCP) including both design and implementation aspects,
- the validation of our prototype implementation performed on multiple platforms including our laboratory testbed, the Emulab network emulation environment [8] and the ns-2 network simulator [9],
- as well as a comprehensive performance evaluation of DFCP in comparison with TCP
  - on both simulation and testbed platforms,
  - on multiple network topologies,
  - by investigating two widely used TCP versions in various network conditions and
  - focusing on important performance metrics such as goodput, flow completion time and fairness.

The paper is organized as follows. First, in Section 2 we give a brief overview about the evolution of TCP and digital fountain codes. We also review the recent ideas proposed in the field of data transfer methods and protocols. In Section 3, we introduce and discuss the envisioned network architecture built upon our newly developed transport protocol. Section 4 gives a detailed description of DFCP including the main design principles and implementation issues. In Section 5, we describe the evaluation methodology used for validation and comparative performance analysis. In Section 6, we present a multi-platform validation framework and confirm the operability of our protocol. A comprehensive performance evaluation study of DFCP and different TCP versions is presented in Section 7, then the future applications and challenges are discussed in Section 8. Finally, Section 9 concludes the paper.

2. Related work

2.1. The evolution of TCP

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 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 by determining the proper sending rate and to achieve high performance. TCP maintains a congestion window that controls the number of outstanding unacknowledged data packets in the network. Continuously evolving network environments have made significant research demand on designing more and more efficient transport protocols in the last decades. As a result, several TCP versions have been developed in order to fit the ever-changing requirements of communication networks.

In this paper, we investigate two widely used TCP variants, namely TCP Cubic [10] and TCP NewReno with SACK [11]. In the case of TCP Reno, a lost packet is detected and retransmitted when triple duplicate acknowledgments are received or a timeout event occurs at the sender. TCP Reno is effective to recover from a single packet loss, but it still suffers from performance problems when multiple packets are dropped from a window of data [12]. TCP NewReno is a slight modification of TCP Reno intended to improve its performance when a burst of packets is lost [11]. TCP Cubic is an enhanced version of its predecessor, BIC TCP [13]. BIC TCP was originally designed to solve the well-known RTT (round-trip time) 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 since the last loss event, which provides good stability and scalability [10].

Beyond the protocols described above, many other solutions have been worked out to improve the performance of traditional 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) [14] was proposed to achieve better performance on high-capacity links by modifying the congestion control mechanism of TCP for use with large congestion windows. Scalable TCP (STCP) [15] applies a multiplicative increase and multiplicative decrease algorithm to obtain performance improvement in high-speed networks and it can also guarantee the scalability of the protocol. TCP Westwood [16] 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 Reno cannot distinguish between random and congestion losses, thus equally reacts to them. In fact, TCP Westwood shows moderate sensitivity to random errors, therefore the improvement is the most significant in wireless networks with lossy links. FAST TCP [17] is a congestion avoidance algorithm especially targeted for long-distance, high-latency links. FAST determines the current congestion window size based on both round-trip delays and packet losses over a path. The algorithm estimates the queuing delay of the path using RTTs 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. MultiPath TCP (MPTCP) [18] is a recent approach for enabling the simultaneous use of several IP addresses or interfaces by a modification of TCP that presents a regular TCP interface to applications, while in fact spreading data across several subflows.

Traditional TCP and its variants were primarily designed for wired networks, but emerging wireless networks and the increasing demands motivated researchers to develop new versions and optimize them for different network environments. 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 various applications as degradation of throughput, inefficiency in network resource utilization and excessive interruption of data transmissions. Modification of standard TCP to remedy its deficiency in wireless

communication has been an active research area in the recent years, and many schemes have been proposed for various environments such as satellite, ad-hoc and cellular networks [19].

2.2. 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 [6]. 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 $n \to \infty$).

Historically, Tornado codes [20] were the first modern fountain codes, but they were proven to be impractical due to the requirement for a cascade of graphs. Hence, they were quickly replaced by irregular Luby Transform (LT) codes [21], which have much simpler structure and equal or even better performance. The main drawback of LT codes was that they failed to provide low complexity encoding and decoding operations.

In these days, Raptor codes [22] are the most efficient ones in the family of fountain codes. They offer linear time encoding and decoding complexity, as well as require only a small number of XOR operations for each generated symbol. Raptor codes are specified in [23], but they have also been adopted into multiple standards, such as in the area of broadcast file delivery and streaming services. Currently, the most flexible and powerful variant of Raptor codes is RaptorQ [24], which supports larger source block sizes and provides better coding efficiency.

2.3. Data transfer paradigms

In the last decade, the issues of TCP motivated researchers to find alternative ways for data transfer beside the traditional congestion control based approach. One of these ideas was suggested by GENI (Global Environment for Network Innovations) [25], recommending the omission of congestion control from the transport layer and promoting erasure coding schemes instead to handle congestion and its consequences. Unfortunately, no realization or further refinement of this concept has been published so far. In the following, we review some related work.

Raghavan and Snoeren proposed a decongestion controller and investigated its properties in [26]. Bonald et al. studied the network behavior in the absence of congestion control [27]. We emphasize their astonishing result, that is, operating a network without congestion control does not necessarily result in congestion collapse.

Furthermore, many research works focused on the application of erasure codes in data transport. López et al. investigated a fountain based protocol using game theory [28]. They found that a Nash equilibrium can be obtained, and at this equilibrium, the performance of the network is close to the performance experienced when all hosts use TCP. Kumar et al. proposed a transport protocol based on fountain codes for wireless networks and pointed out that, regarding performance, this approach can be beneficial in such environments [29]. Botos et al. suggested a modified TCP for high loss rate environment by utilizing rateless erasure codes [30]. In their proposal, the well-known slow-start and congestion avoidance algorithms of TCP are used, but some modifications are suggested to avoid the dramatic decrease of the sending rate in case of high packet loss. Another mechanism called TCP/NC that incorporates network coding into TCP with only minor changes to the protocol stack is presented in [31]. 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 congestion control in lossy environments such as wireless networks. Cui et al. proposed FMTCP (Fountain code-based Multipath TCP) in [32] to improve the performance of MPTCP by effectively mitigating the negative impact of the heterogeneity of different paths. FMTCP takes advantage of the random nature of fountain codes to flexibly transmit encoded symbols from the same or different data blocks over different subflows, and also achieves much more stable performance under abrupt changes of path quality.

3. Networking without congestion control

This section is devoted to envision a network architecture based on digital fountain based transfer and to highlight the benefits of this approach with potential future applications. The main component of the architecture is a novel 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.

3.1. 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 [22] 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.

The suggested network architecture relying on *digital fountain based error correction* is shown in Fig. 1. 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) [33], 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, as fairness can be treated in an orthogonal way. The feasibility of this approach is supported by the scalability of per-flow fair queuing [34,35].

3.2. 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*, 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

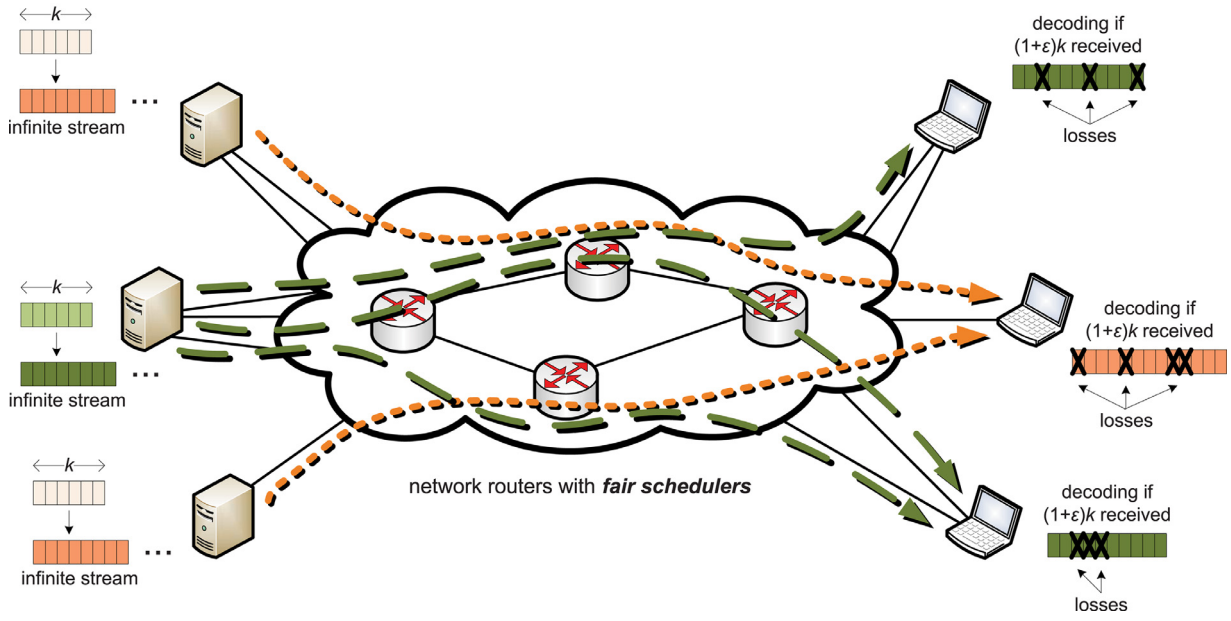


Fig. 1. The network architecture utilizing digital fountain based data transfer.

the case of UDT (UDP-based Data Transport) [36]. Available bandwidth estimation (ABwE) [37] 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. The majority of ABwE 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, IGI, 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. Different ABwE 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 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 (overestimation is preferred), 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*, e.g. as XCP (eXplicit Control Protocol) [38] does. One of the most widely known solutions is called ECN (Explicit Congestion Notification) [39], which allows to signal congestion by marking packets instead of dropping them from the buffer. The re-ECN [40] 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) [41] 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 [42] 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.

3.3. Potential benefits

In this part, we give some possible applications and use-cases of the proposed networking paradigm. 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*

Fig. 2. Protocol header structure.

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 8.

4. DFCP: design and implementation

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 currently under research and development, we close this section with adjustable protocol specific parameters intended to facilitate future experimentations.

4.1. 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 an extra overhead, it will be shown in Section 7 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.

4.2. Protocol header

The protocol header can be seen in Fig. 2 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 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. The checksum is a generated number depending on the content of the header and partially on the data field.

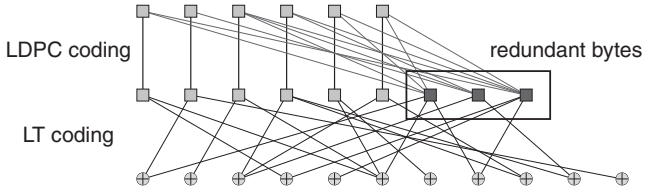


Fig. 3. Encoding phases of message blocks.

4.3. Connection establishment and signaling

DFCP's connection establishment is based on a three-way handshake procedure as in the case of TCP [43]. 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. 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.

4.4. Coding scheme

The flow chart of the coding and data transfer process can be seen in Fig. 4. 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 Fig. 3, Raptor coding [22] involves two phases: *precoding* and *LT coding* [21]. In our implementation, precoding is realized by LDPC (Low-Density Parity-Check) coding [44], 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.

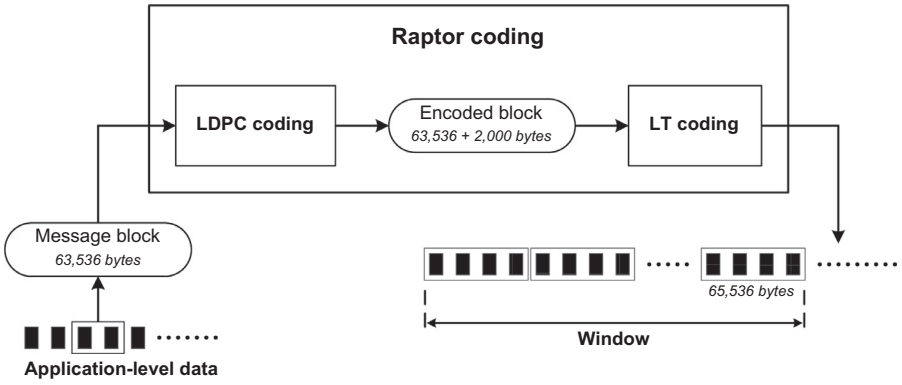


Fig. 4. The flow chart of the coding and data transfer process.

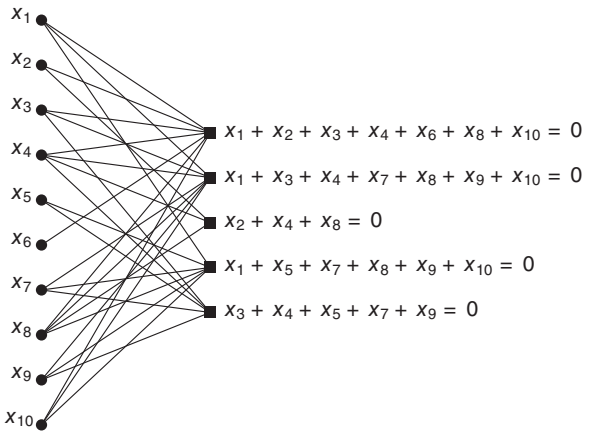


Fig. 5. Example of an LDPC code.

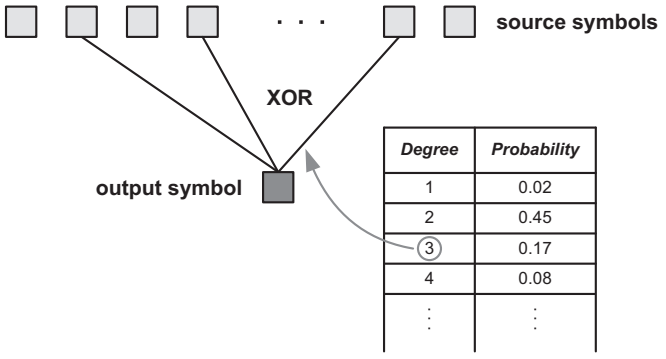


Fig. 6. The concept of LT coding.

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 *messages nodes* and *check nodes*, respectively. An example is shown in Fig. 5. 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 by using 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:

$$c_r = c_r \oplus m_i \tag{1}$$

where c_r denotes the randomly chosen check node and m_i is the actual message node. 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 $n - k$ 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.

The second phase of the Raptor coding scheme, *LT coding*, is performed on an encoded block of 65536 bytes received from the LDPC encoder. Fig. 6 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.

4.5. 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 Fig. 4). 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, first of all 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 [24], which is currently the most efficient variant of Raptor codes, only two additional symbols can provide a successful decoding probability greater than 99.9999%. Decoding failure is very rare, but when it occurs the extra packets received in the meantime will be enough for a successful outcome. 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). 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.

4.6. Main parameters

Since DFCP is currently under development, it is important to make it possible to experiment by adjusting some protocol specific parameters. In the recent version of the protocol, the following parameters can be set:

- **Window size.** It controls the 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 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.
- **Acknowledgments.** ACKs can be switched ON or OFF that is advantageous for experimental reasons. The purpose of using acknowledgments is twofold: (1) it gives a feedback to the sender about the blocks successfully received by the destination host, and (2) controls the speed of the sender preventing buffer overflow at the receiver side. In OFF state, we can investigate the properties of the maximal rate sending mechanism by ignoring many subsidiary factors.
- **Encoding and decoding.** Coding phases can be switched ON or OFF independently of each other to study the impact of the Raptor codec implementation on the performance of DFCP. If encoding is set to OFF, only the first block is encoded making possible to ignore the overhead of the encoding process. In this case, all message blocks are replaced by this block and it is sent instead of the original blocks. If decoding is switched OFF, decoding is not performed, but it can be determined by the receiver that successful decoding would be possible or not. This option enables to separate the coding process from the transport mechanism, and hence, to focus on the new data transfer paradigm. The use of Raptor codes is only one possible option for encoding data, and our concept is not restricted to the type of fountain code and is open for its future evolution.

5. Performance 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 [45,46]. 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 very difficult to repeat [47,48].

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 [9]. 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 settings and parameters used in the analysis.

5.1. Performance metrics

To evaluate the performance of transport protocols, there are some well-known metrics. One of the most widely used measures is *throughput*, which gives the amount of data successfully transferred per second from source to destination. 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 taking into account 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 (Fig. 7) 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

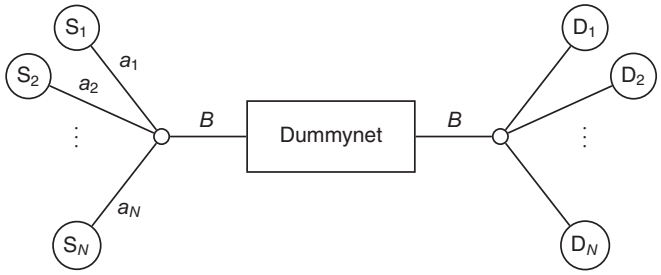


Fig. 7. Dumbbell topology with N source-destination pairs.

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 (I is an indicator variable):

$$G_i = \begin{cases} \dfrac{c_B}{(1+\varepsilon_i)N} & \forall j : c_j \geq \frac{c_B}{N} \\ \dfrac{c_i}{1+\varepsilon_i} & c_i < \frac{c_B}{N} \\ \dfrac{c_B - \sum_{k=1}^{N} I_{\{c_k < \frac{c_B}{N}\}} c_k}{(1+\varepsilon_i)\sum_{k=1}^{N} I_{\{c_k \geq \frac{c_B}{N}\}}} & \exists j : c_j < \frac{c_B}{N} \text{ and } c_i \geq \frac{c_B}{N} \end{cases}. \quad (2)$$

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 [49]. 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 [50], 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. In our experiments we used the Jain's index as the fairness measure, which is a widely accepted fairness index in the literature [51]. Jain's index can be calculated by the following formula:

$$JI = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2} \quad (3)$$

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

5.2. 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 [52]. The dumbbell topology consisting of N source-destination pairs can be seen in Fig. 7. First, we experimented with a single flow ($N = 1$) to reveal the ability of DFCP to resist against varying *delay* and *packet loss rate* parameters of the network. 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 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).

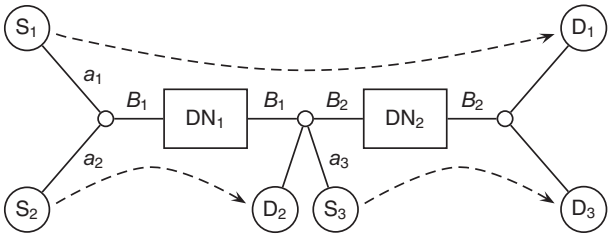


Fig. 8. Parking lot topology with three source-destination pairs.

The scenarios described above made possible to explore the fundamental features of DFCP and its scalability. Beyond these experiments, DFCP was studied in a more realistic environment as well. Fig. 8 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 sections.

Measurements lasted for 60 s in most scenarios (except if mentioned otherwise), and the results were obtained by excluding the first 15 s in order to ignore the impact of transient behavior of the investigated transport protocols. In the case of multiple flows they were started at the same time, and for scheduling discipline WFQ (Weighted Fair Queuing) was applied by default with equal weights [53]. However, we also experimented with other fair schedulers like DRR (suggested previously for our paradigm) [33] and SFQ (Stochastic Fair Queuing) [54], as well as with FIFO scheduler (using the DropTail queue management policy) which is the simplest algorithm available in today's network routers.

5.3. Test environments

To validate the performance evaluation results, the test scenarios were executed on the following three different platforms independently [55], and here we give a brief description of each including

(1) our laboratory testbed,
(2) the Emulab network emulation environment,
(3) as well as the ns-2 network simulator.

The *laboratory testbed* consisted of senders, receivers and a Dummynet network emulator [56], 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 1.

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 measurement setup 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 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. Similar to the testbed measurements, our modified kernel including the implementation of DFCP was loaded into the test computers.

Table 1
Hardware components of our laboratory test computers.

Component	Type and parameters
(a) Hardware components of senders and receivers	
Processor	Intel® Core™2 Duo E8400 @ 3 GHz
Memory	2GB DDR2 RAM
Network adapter	TP-Link TG-3468 Gigabit PCI-E
Operating system	Debian Lenny with modified kernel
(b) Hardware components of network emulators	
Processor	Intel® Core™ i3-530 @ 2.93 GHz
Memory	2GB DDR2 RAM
Network adapter	TP-Link TG-3468 Gigabit PCI-E
Operating system	FreeBSD 8.2

Table 2
Hardware components of the Emulab test computers.

Component	Type and parameters
(a) Hardware components of senders and receivers	
Processor	Intel® Xeon® processors @ 3 GHz
Memory	2GB DDR2 RAM
Network adapter	Intel Gigabit PCI-E
Operating system	Debian Lenny with modified kernel
(b) Hardware components of network emulators	
Processor	Intel® Xeon® E5530 @ 2.40 GHz
Memory	12GB DDR2 RAM
Network adapter	Broadcom NetXtreme II 5709 Gigabit PCI-E
Operating system	FreeBSD 8.3

The third tool was the *ns-2 network simulator* to validate DFCP, which is widely used by researchers to try out and evaluate their new methods [9]. Since the first prototype of DFCP has been implemented in the Linux kernel, we had to find a way to simulate our protocol 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 [57]. Focusing on these requirements, we chose *Network Simulation Cradle (NSC)*, which is a framework for wrapping kernel code into simulators allowing the simulation of real-world behavior at little extra cost [58]. NSC supports the simulation of the network stacks of many operating systems such as FreeBSD, OpenBSD, lwIP and Linux. This tool has been validated by comparing situations using a test network with the same situations in the simulator, and it has been shown that NSC is able to produce extremely accurate results. Moreover, it has been ported to several network simulators including both ns-2 and ns-3. Although, NSC is an excellent tool for simulating different TCP versions and new TCP-like transport protocols, we had to carry out a challenging work to get NSC able to handle DFCP, which is based on a novel paradigm and it is significantly different compared to the principles applied by TCP.

6. Protocol validation

In this section, we present a validation study carried out on the three different testing platforms discussed above. The results show that DFCP performs in a similar way in these environments providing a strong evidence for the operability of our protocol. We also quantify the impact of the main parameters of DFCP on the goodput performance.

6.1. Operation in different network conditions

Table 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 Fig. 7). 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 and it can achieve almost the same goodput for different RTT values.

Table 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

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

6.2. The impact of protocol specific parameters

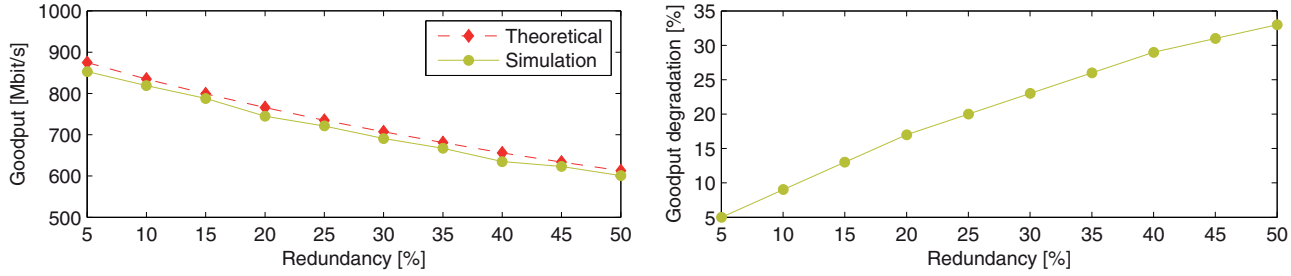
Redundancy, denoted by ϵ, highly determines the efficiency of fountain coding schemes since a lower value makes it possible to transmit more useful data bytes at a given link. Fig. 9 demonstrates how the redundancy parameter of DFCP affects the goodput performance when the window size is set to 1000 blocks. The theoretical curve of Fig. 9a is derived from the goodput formula (2) defined in Section 5 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. Fig. 9b 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 for useful data transmission. In practice, the typical value of redundancy is below 5% for recent fountain codes [22].

Fig. 10 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 4, 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.

7. Comprehensive performance evaluation

In this section, we present an extensive performance analysis study by comparing DFCP to different TCP versions, namely TCP Cubic [10] which is the default congestion control algorithm in the Linux kernel and TCP NewReno with SACK option [11]. In the following, we deal with five main aspects:

(1) goodput performance,
(2) buffer demand and occupancy,
(3) flow transfer efficiency,



(a) Goodput performance for increasing redundancy

(b) Performance degradation for increasing redundancy

Fig. 9. The impact of the redundancy parameter.

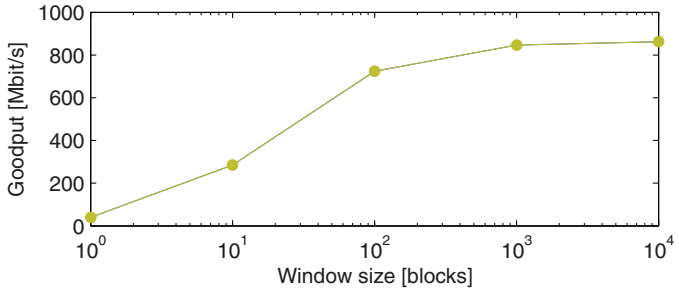


Fig. 10. The impact of window size on the goodput performance.

(4) fairness properties,
(5) as well as scalability.

Moreover, we also quantify the bandwidth wasted by our data transfer mechanism due to the dead packet phenomenon if an SDN-based rate control algorithm is employed. All measurements were performed on three testing platforms (testbed, Emulab and ns-2), and we present a selection of these 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 scalability measurements.

7.1. 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 Fig. 11. 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. 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.

Fig. 12 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-

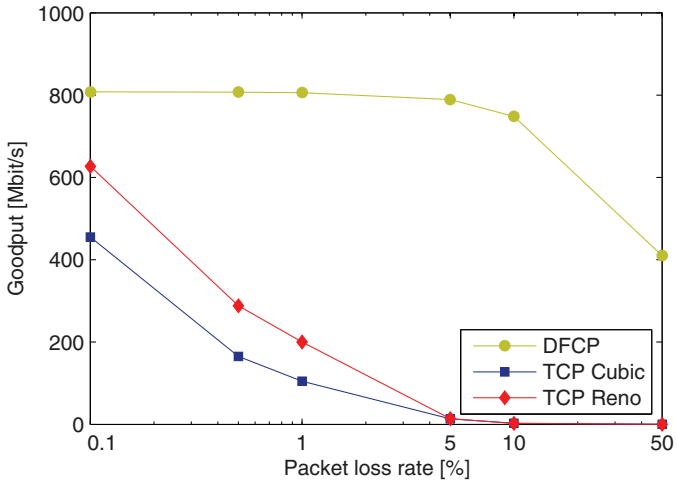


Fig. 11. The performance of DFCP and TCPs in a lossy environment (simulation).

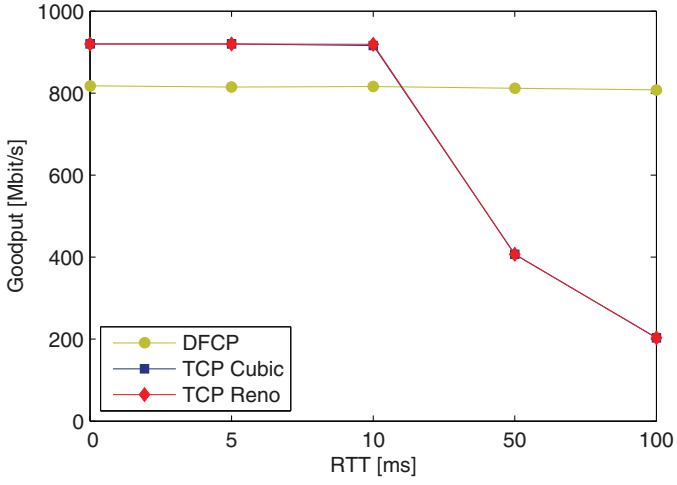


Fig. 12. The performance of DFCP and TCPs for varying RTT (simulation).

trip time in a real network exceeds 10 ms [59], DFCP can function more efficiently than TCP in such conditions.

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

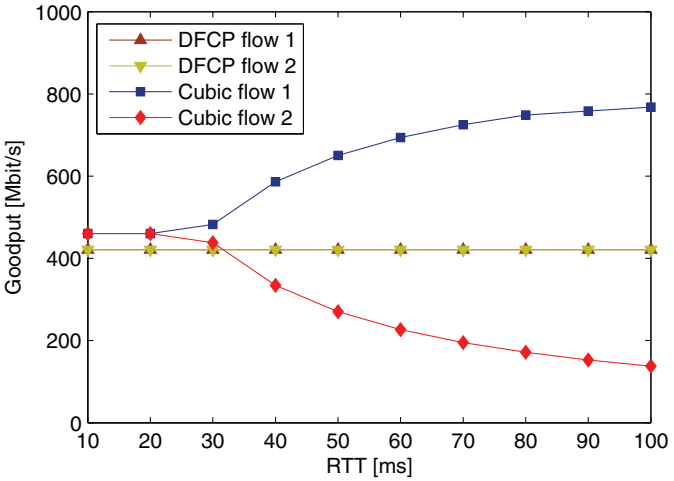


Fig. 13. The performance of DFCP and TCP in the case of two competing flows (testbed).

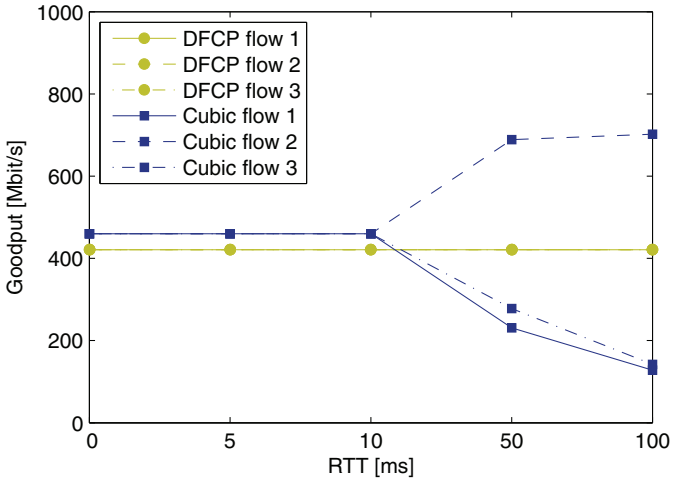
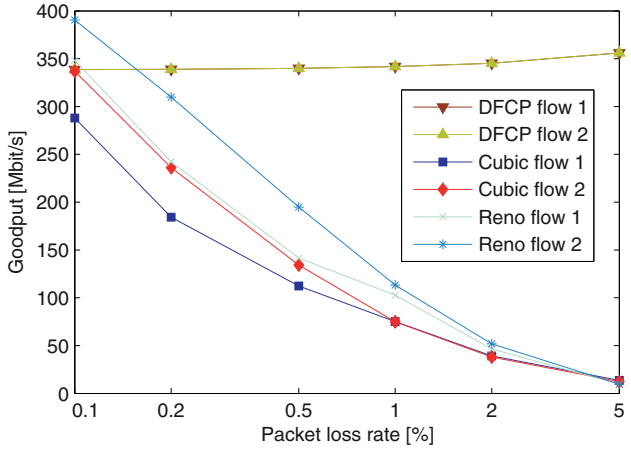


Fig. 15. The behavior of DFCP and TCP in a multi-bottleneck network with varying delay (testbed).

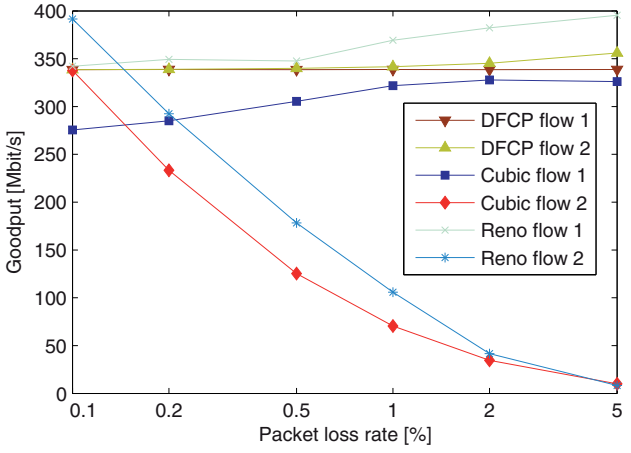
known, standard TCP cannot provide an equal portion of the bottleneck bandwidth for competing flows with different round-trip times [60] due to its AIMD mechanism [51]. Fig. 13 depicts the goodput for two competing DFCP and TCP Cubic flows, respectively. The delay of *flow 1* was fixed at 10 ms, and 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 unfairness 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.

Fig. 14 illustrates the impact of packet loss rate on the goodput performance for two competing flows. Fig. 14a shows the case when packet loss rates are equal for both flows and changed according to the horizontal axis, and Fig. 14b shows the case when they experienced different loss rates. In the latter case, the first flow has a fixed loss rate set to 0.1%, and the second one has a loss rate varied between 0.1 and 5% as shown in Fig. 14a. Fig. 14a, it can be observed that both TCP Cubic and TCP Reno flows do not 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 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. Fig. 14b 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.

Fig. 15 presents the performance comparison of DFCP and TCP Cubic carried out on the parking lot topology illustrated in Fig. 8 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



(a) Equal packet loss rate



(b) Different packet loss rates

Fig. 14. Goodput for two competing flows with equal and different packet loss rates (testbed).

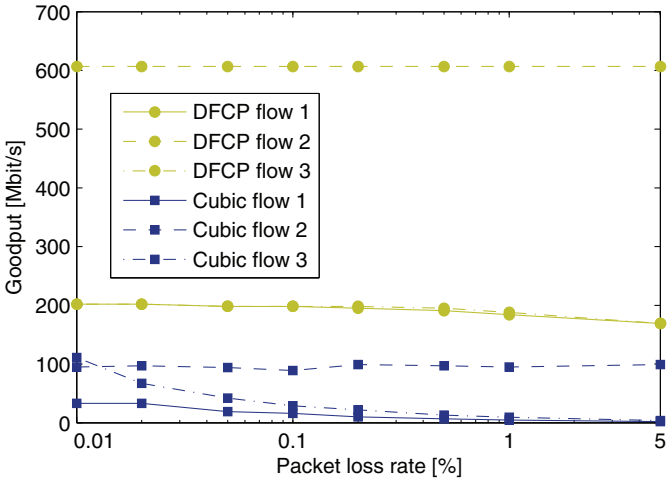


Fig. 16. The behavior of DFCP and TCP in a multi-bottleneck network with varying packet loss rate (testbed).

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 and more bandwidth. Accordingly, TCP Cubic does not provide fairness between *flow 1* and *flow 2* having different RTTs. Moreover, in this case 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.

Fig. 16 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 TCP variants is quite poor due to their sensitivity to this factor.

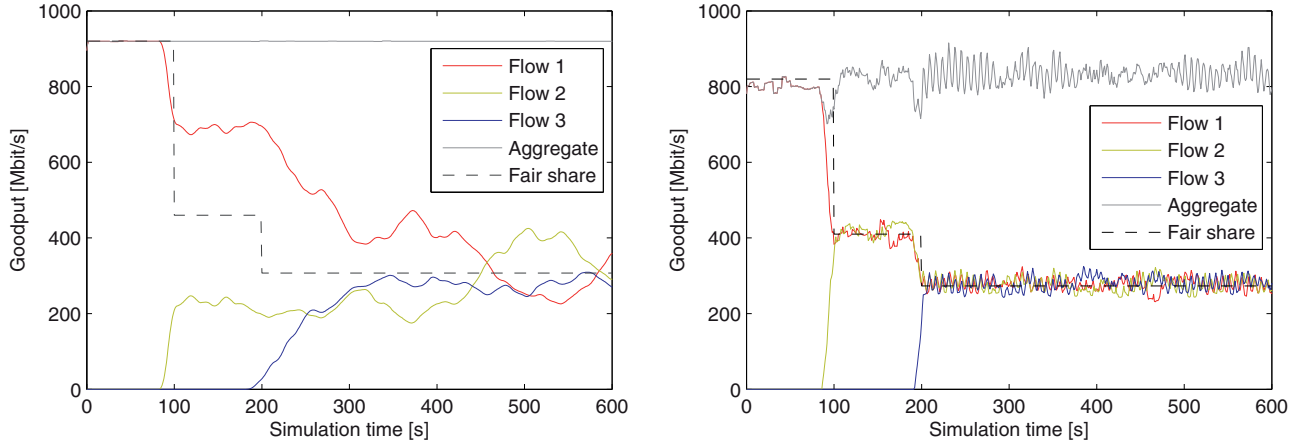
In Fig. 17, the dynamics of three concurrent flows is illustrated when each flow was started at different times. This scenario was carried out on the dumbbell topology where we investigated the two data transfer paradigms, TCP Cubic with FIFO (see Fig. 17a) and DFCP with DRR scheduling (see Fig. 17b). The bottleneck link capacity and the round-trip delay were set to 1 Gbps and 50 ms, respectively. The buffer size was equal to the BDP (bandwidth-delay product), and the flows were started at 0, 100 and 200 s. The goodput curve was smoothed by using a 10 s long moving window. We can observe that TCP flows show a slow convergence to the fair share and then their goodput highly fluctuates around it. In the case of DFCP the convergence time is very low while the fluctuation around the fair share remains moderate. However, the transfer mechanism of DFCP leads to a more bursty transmission than that of TCP, which is due to the trade-off between the window size and the burstiness of traffic as pointed out in Section 6. An extensive analysis of the dynamic behavior of different transport protocols can be found in [61].

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. The results suggest that the possible application areas cover both high latency and high loss rate network environments.

7.2. 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 [62]. 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 [63].

Fig. 18 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



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

Fig. 17. Flow dynamics in the case of three concurrent flows started with different timings (simulation).

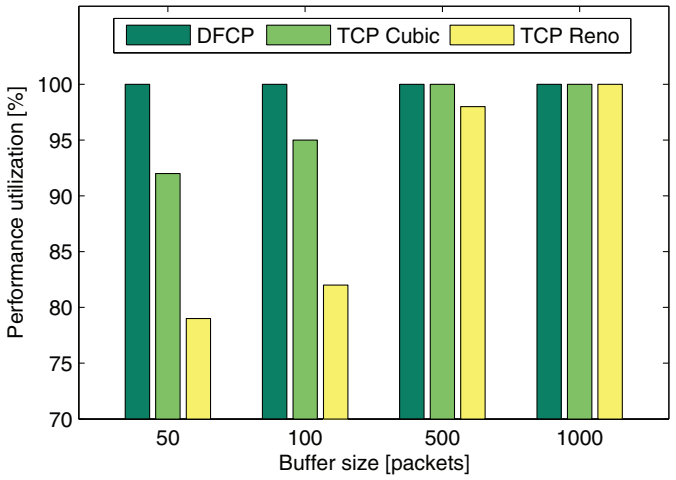


Fig. 18. The impact of buffer size on the performance of DFCP and TCPs (simulation).

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 [63].

Fig. 19 illustrates how DFCP and different TCP versions utilize the bottleneck buffer. The average occupancy was calculated for a 600 s long interval. In the case of a small buffer of 100 packets (see Fig. 19a), 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 Fig. 19b), 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 its size. TCP Cubic and Reno also show an improvement in utilization as the average occupancy is 75% and 58%, respectively.

Our results revealed the robust property of DFCP regarding 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.

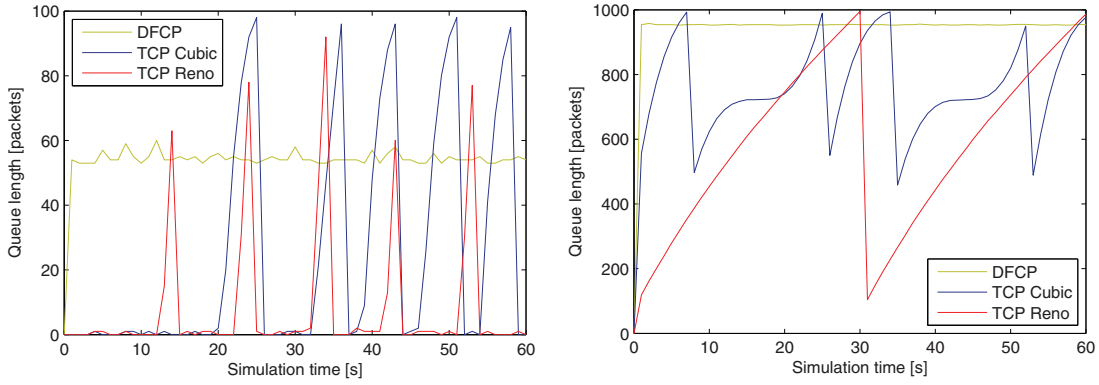
7.3. Flow transfer efficiency

As we mentioned in Section 5, *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 [64]. Accordingly, we investigated two different categories: (1) web object (150 kB, the mean size is about 100–200 kB [65]) and (2) DVD (4.7GB), which represent short and long data transfers, respectively.

Fig. 20 illustrates how the flow completion time depends on the packet loss rate. The flow completion times longer than 60 s were calculated by using the steady-state goodput for each figure of this section. One can see that in both cases DFCP provides the fastest download indicating its potential 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 Fig. 20a). However, if we transfer a full DVD, the advantage of DFCP is pronounced in the whole range of packet loss rate (see Fig. 20b). 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 Fig. 21. Specifically, in the case of a web object there are several orders of magnitude between the download time of DFCP and TCP for increasing round-trip time (see Fig. 21a). Considering the category of DVD, it can be stated that the difference in download time is negligible for low RTT values, however, it gets more and more significant towards high RTT values as shown in Fig. 21b.

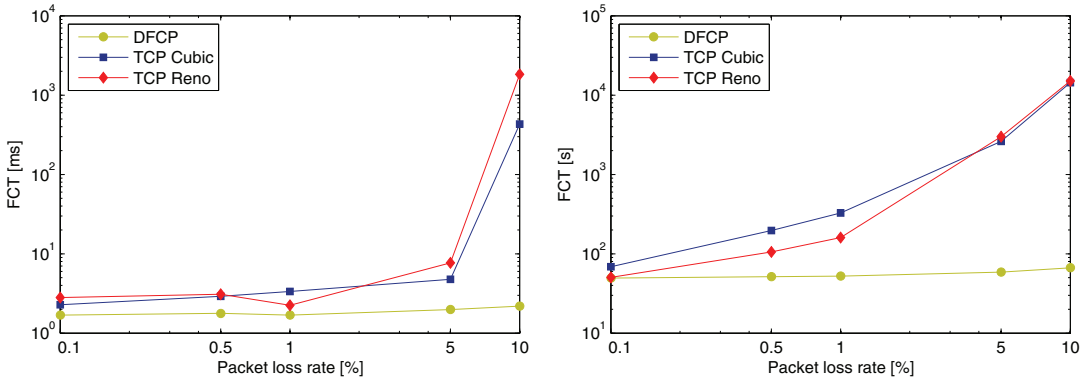
Fig. 22 shows the flow completion time for two competing DFCP and TCP Cubic flows where the first flow has a fixed RTT of 10 ms and the delay of the second flow is varied between 10 and 100 ms. We observed that the results for TCP Reno were quite the same as in the case of TCP Cubic, hence only the latter was depicted. Looking at Fig. 22a, 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 Fig. 22b). In contrast, DFCP flows attain equal download time in the



(a) Buffer size: 100 packets

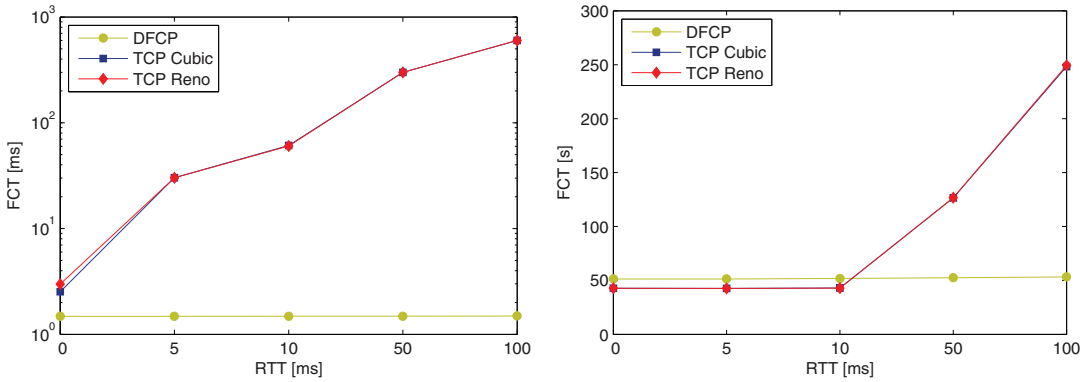
(b) Buffer size: 1000 packets

Fig. 19. Buffer occupancy (simulation).



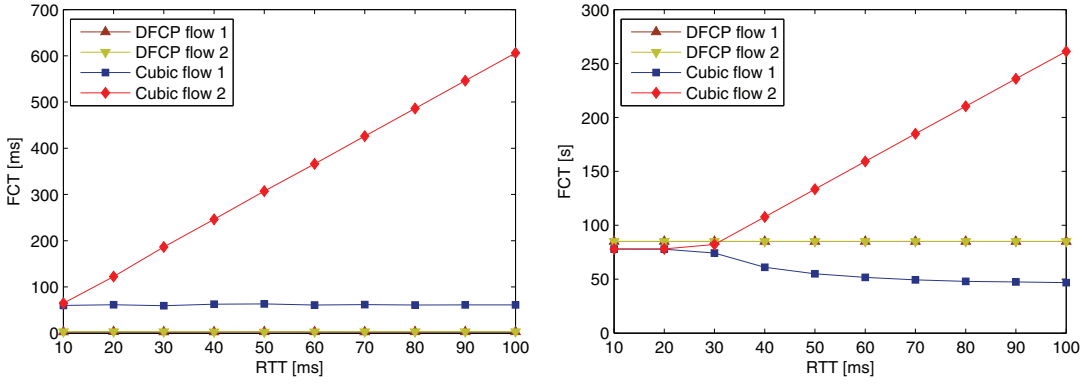
(a) Web object (b) Full DVD

Fig. 20. Flow completion time for different packet loss rates (testbed).



(a) Web object (b) Full DVD

Fig. 21. Flow completion time for different round-trip times (testbed).



(a) Web object (b) Full DVD

Fig. 22. 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).

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 section. 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 a much more effective transport mechanism like DFCP.

7.4. Fairness with different queuing mechanisms

In our proposed future network architecture, *fairness* can be realized by the application of fair schedulers as we mentioned in Section 3. 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. So, the natural question is which one to choose? On the other hand, in most routers a FIFO scheduler is applied by default

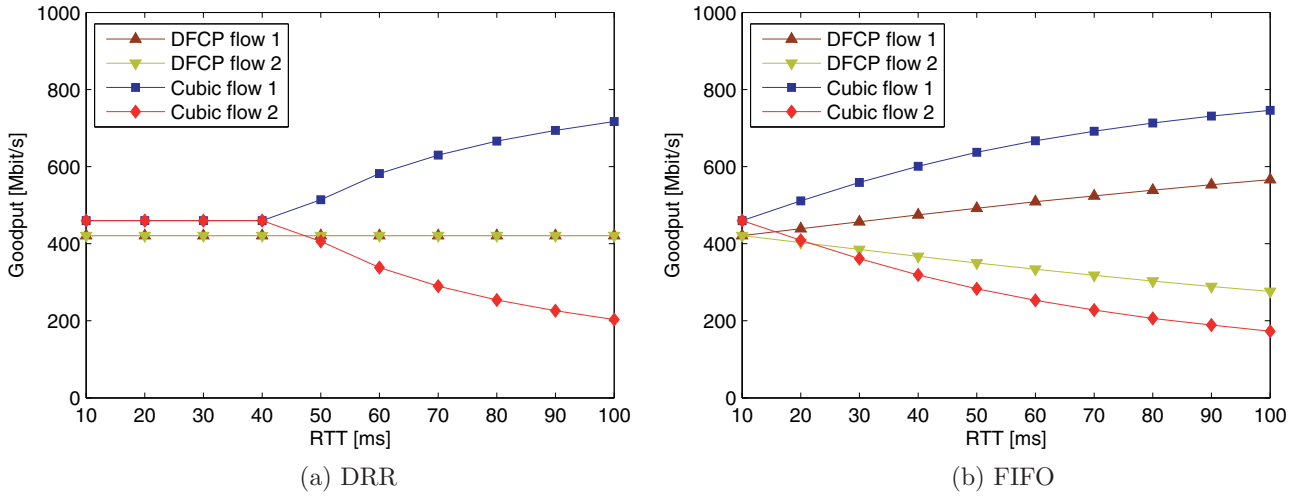


Fig. 23. Bandwidth sharing with different queuing mechanisms (simulation).

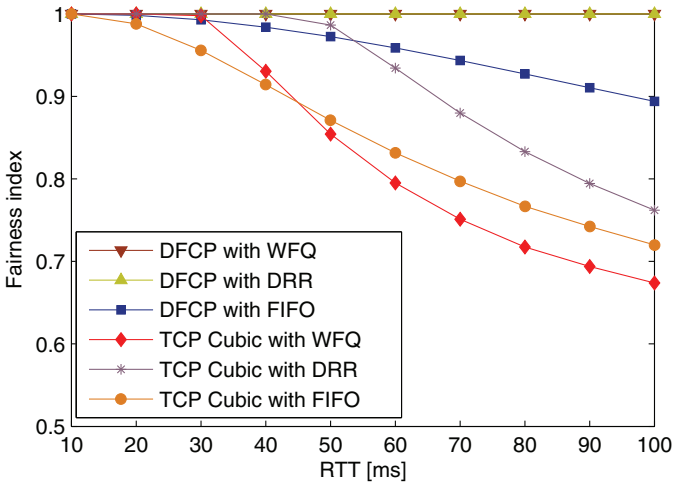


Fig. 24. Intra-protocol fairness with WFQ, DRR and FIFO schedulers (simulation).

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, in this section we extend our fairness analysis by investigating other queuing mechanisms than WFQ. We note that, since the results obtained for SFQ were completely similar to the ones obtained for WFQ, we focus on DRR and FIFO below.

Fig. 23 shows the goodput performance of DFCP and TCP versions for the same test scenario presented in Section 7.1. Fig. 23a indicates that, with DRR scheduler, DFCP can provide equal bandwidth sharing for the competing flows similar to the case 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. Fig. 23b demonstrates that the phenomenon of unfairness 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.

Fig. 24 presents a fairness comparison of different schedulers for DFCP and TCP Cubic. 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 even with the much simpler FIFO algorithm.

Table 4
Performance scalability (simulation).

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

Concluding the observations, we can say 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 (e.g. DRR) in network routers, which can significantly improve TCP-based bandwidth sharing.

7.5. 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 communication 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, TCP Cubic with FIFO (current Internet) and DFCP with DRR scheduling (our concept). The results obtained for a 200 s long measurement period on the topology of Fig. 7 with a 0.1 BDP buffer, and each flow experienced 100 ms of RTT.

Table 4 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 TCP Cubic and DFCP, 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 goodput increases with the number of flows, but decreases with the link capacity. For example, in the

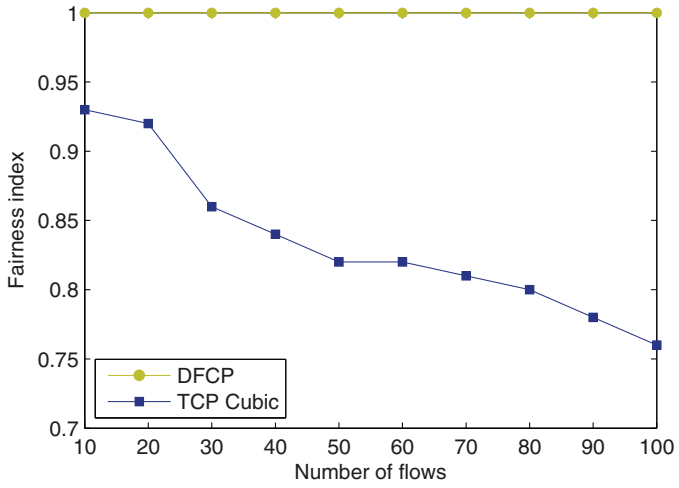


Fig. 25. Fairness for increasing number of competing flows (simulation).

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 Fig. 12), and hence the underutilization of the 10 Gbps link by 10 flows.

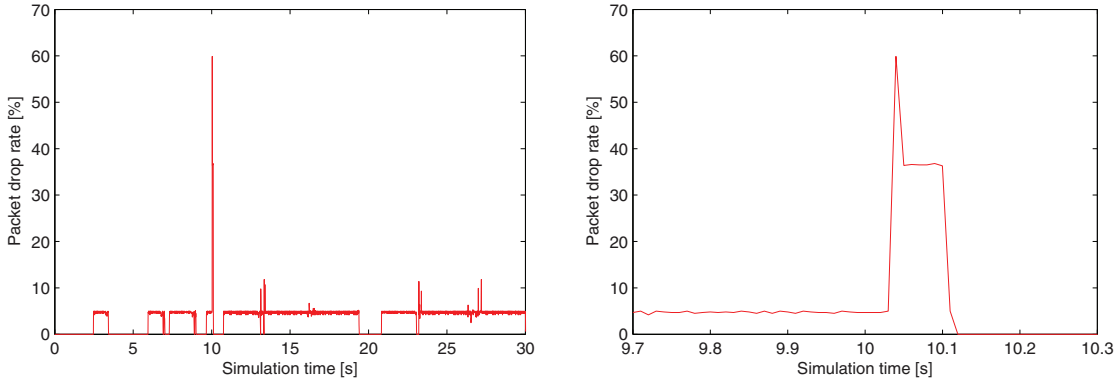
Fig. 25 demonstrates the fairness scalability of DFCP and TCP Cubic with increasing number of flows. In this scenario, each flow experienced the same delay to avoid the phenomenon of RTT unfairness. In spite of that the tendency is obvious for TCP Cubic: the larger the number of concurrent flows, the 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.

7.6. Bandwidth waste

In Section 3.2, we pointed out that our mechanism requires the proper control of transmission rate in order to cope with the so-called dead packets, and hence, to minimize the extent of bandwidth waste. This section 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 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 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 [66] 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 [67] can deliver highly accurate information about the network in real-time without incurring significant overhead whereas OpenNetMon [68] exploits OpenFlow to provide per-flow metrics including throughput, delay and packet loss. Authors in [69] 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 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 Fig. 8 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 s 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. Fig. 26 shows the packet drop rate at the second bottleneck router in the function of time for 5% estimation error and 50 ms response



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

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

Fig. 26. Packet drop rate at the bottleneck router using SDN-driven rate control (simulation).

Table 5
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%

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 s, the available bandwidth on the path of *flow 1* decreases to 200 Mbps since the DRR 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 5 summarizes the mean drop rate calculated over a 30 s 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.

8. Future applications and challenges

Our envisioned architecture built on the transport mechanism of DFCP is a good candidate for data communication of future networks since it is capable of supporting novel applications and use-cases. However, current DFCP implementation has some issues and limitations that need to be solved. In this section, we discuss the potential application areas and future challenges.

Multipath transport has received significant attention in recent years. As a result of these activities, MultiPath TCP (MPTCP) has been standardized by IETF [18]. Moreover, now it is available as a kernel implementation to Linux [70] 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 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 [71]. 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 flows. High-performance storage systems of data centers can also benefit from fountain coding, which enables better utilization of resources by efficient distribution of I/O requests [72].

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. 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 (FEC) and automatic repeat request (ARQ). 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 achieves only 1 percent. In contrast, DFCP is insensitive to packet loss in a wide range of packet loss rates as it was demonstrated in Section 7. 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 proposed transport protocol also has a high potential to deploy it in *optical networks*. This is due to the attractive feature of DFCP concept that it makes possible to build a bufferless network architecture. Since the concept of 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. So the big challenge currently preventing the deployment of building all-optical cross-connects in optical networks can be solved. 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. Another consequence of this vision is that the extra queuing delay in router buffers, which is a significant, hardly estimated and highly variable performance determining factor in our current Internet, can be avoided resulting in an easier network design and dimensioning process.

Finally, we give a summary about the challenges and future plans related to our data transfer paradigm that still need further research. The most important unsolved problem 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* [27]. In fact, it is due to the absence of congestion control, however, there are several possible ways to tackle this issue as discussed in Section 3, and we are currently working on such a mechanism. Another interesting research direction is extending our transport protocol with the capability of *adaptive parameter optimization* during the communication. Considering the *deployment options*, as we pointed out in Section 7.6, SDN is a very attractive environment for DFCP. 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. 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 DFCP 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 infrastruc-

ture. More precisely, physical resources such as link capacities and router memories can be split between the traditional TCP-based and the proposed DFCP-based architectures. During a transition period, it would be set that a given ratio of link capacity and buffer space (e.g. 30%) is maintained for DFCP traffic and the rest is for TCP traffic.

9. Conclusion

In this paper, we advocated a networking paradigm where the objective is not to control congestion but rather to utilize it. In contrast to the present TCP-based Internet, we proposed an architecture built upon a completely different principle by omitting congestion control and applying fair schedulers in network routers. We have designed, developed and implemented a novel transport protocol called Digital Fountain based Communication Protocol (DFCP), which relies on a fountain code based data transfer mechanism. We validated the performance of DFCP on various network topologies and on multiple platforms including our laboratory testbed, the Emulab network emulation environment and the ns-2 network simulator. Moreover, we carried out a comparative performance evaluation of DFCP with the most relevant TCP variants. We found that unlike TCP versions, DFCP is insensitive to packet loss and delay in a wide range of realistic working regimes highlighting its benefits for many areas like wireless communication. In addition, we concluded from the results that DFCP is able to work with small buffers, hence it can support bufferless all-optical networking. From the Quality of Experience (QoE) point of view, we investigated the flow transfer efficiency for both short-lived and long-lived flows, and pointed out that the user experience can be significantly improved by using DFCP. We also showed that digital fountain based transport guarantees good scalability both in terms of performance and fairness for increasing number of flows and link capacity. Finally, we discussed the possible applications of our proposal and outlined some open issues.

References

[1] V. Jacobson, Congestion avoidance and control, in: Proceedings of the 1988 ACM SIGCOMM Symposium, Stanford, CA, USA, 1988, pp. 314–329.
[2] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, Host-to-host congestion control for TCP, IEEE Commun. Surv. Tutor. 12 (3) (2010) 304–342.
[3] Y.T. Li, D. Leith, R.N. Shorten, Experimental evaluation of TCP protocols for high-speed networks, IEEE/ACM Trans. Netw. 15 (5) (2007) 1109–1122.
[4] S. Molnár, B. Sonkoly, T.A. Trinh, A comprehensive TCP fairness analysis in high speed networks, Comput. Commun. 32 (13-14) (2009) 1460–1484.
[5] 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 Workshops, Macau, China, 2012, pp. 222–230.
[6] D.J.C. MacKay, Fountain codes, IEE Proc. Commun. 152 (6) (2005) 1062–1068.
[7] S. Molnár, Z. Móczár, A. Temesváry, B. Sonkoly, Sz. Solymos, T. Csicsics, Data transfer paradigms for future networks: fountain coding or congestion control? in: Proceedings of the IFIP Networking 2013 Conference, New York, NY, USA, 2013, pp. 1–9.
[8] Emulab Network Emulation Testbed, http://www.emulab.net/.
[9] ns-2 Network Simulator, http://www.isi.edu/nsnam/ns/.
[10] I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, in: Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks, Lyon, France, 2005, pp. 1–6.
[11] S. Floyd, T. Henderson, A. Gurtov, The newreno modification to TCP's fast recovery algorithm, RFC 3782, IETF, 2004.
[12] K. Fall, S. Floyd, Simulation-based comparisons of tahoe, reno, and SACK TCP, ACM SIGCOMM Comput. Commun. Rev. 26 (3) (1996) 5–21.
[13] L. Xu, K. Harfoush, I. Rhee, Binary increase congestion control (BIC) for fast long-distance networks, in: Proceedings of the 23rd IEEE International Conference on Computer Communications, Hong Kong, China, vol. 4, 2004, pp. 2514–2524.
[14] S. Floyd, Highspeed TCP for large congestion windows, RFC 3649, IETF (2003).
[15] T. Kelly, Scalable TCP: Improving performance in high-speed wide area networks, ACM Comput. Commun. Rev. 33 (2) (2003) 83–91.
[16] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, R. Wang, TCP westwood: end-to-end congestion control for wired/wireless networks, ACM J. Wirel. Netw. 8 (5) (2002) 467–479.
[17] C. Jin, D.X. Wei, S.H. Low, FAST TCP: motivation, architecture, algorithms, performance, IEEE/ACM Trans. Netw. 14 (6) (2006) 1246–1259.
[18] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, Architectural guidelines for multipath TCP development, RFC 6182, IETF (2011).
[19] Y. Tian, K. Xu, N. Ansari, TCP in wireless environments: problems and solutions, IEEE Commun. Mag. 43 (3) (2005) 27–32.
[20] M. Luby, Tornado codes: practical erasure codes based on random irregular graphs, in: Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science, Barcelona, Spain, 1998, pp. 171–175.
[21] M. Luby, LT codes, in: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002, pp. 271–280.
[22] A. Shokrollahi, Raptor codes, IEEE Trans. Inf. Theory 52 (6) (2006) 2551–2567.
[23] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, Raptor forward error correction scheme for object delivery, RFC 5053, IETF (2007).
[24] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder, Raptorq forward error correction scheme for object delivery, RFC 6330, IETF (2011).
[25] D. Clark, S. Shenker, A. Falk, GENI Research Plan (Version 4.5), 2007.
[26] B. Raghavan, A.C. Snoeren, Decongestion control, in: Proceedings of the 5th ACM Workshop on Hot Topics in Networks, Irvine, CA, USA, 2006, pp. 61–66.
[27] T. Bonald, M. Feuillet, A. Proutiere, Is the 'law of the jungle' sustainable for the internet? in: Proceedings of the 28th IEEE Conference on Computer Communications, Rio de Janeiro, Brazil, 2009, pp. 28–36.
[28] L. López, A. Fernández, V. Cholvi, A game theoretic comparison of TCP and digital fountain based protocols, Comput. Netw. 51 (12) (2007) 3413–3426.
[29] D. Kumar, T. Chahed, E. Altman, Analysis of a fountain codes based transport in an 802.11 WLAN cell, in: Proceedings of the 21st International Teletraffic Congress, Paris, France, 2009, pp. 1–8.
[30] A. Botos, Z.A. Polgar, V. Bota, Analysis of a transport protocol based on rateless erasure correcting codes, in: Proceedings of the 2010 IEEE International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, vol. 1, 2010, pp. 465–471.
[31] J.K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, J. Barros, Network coding meets TCP: theory and implementation, Proc. IEEE 99 (3) (2011) 490–512.
[32] 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, Macau, China, 2012, pp. 366–375.
[33] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round-robin, IEEE/ACM Trans. Netw. 4 (3) (1996) 375–385.
[34] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, On the scalability of fair queuing, in: Proceedings of the 3rd ACM Workshop on Hot Topics in Networks, San Diego, CA, USA, 2004, pp. 1–6.
[35] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing, in: Proceedings of the ACM SIGMETRICS 2005 International Conference on Measurement and Modeling of Computer Systems, Banff, AB, Canada, 2005, pp. 217–228.
[36] Y. Gu, X. Hong, R.L. Grossman, Experiences in design and implementation of a high performance transport protocol, in: Proceedings of the 2004 ACM/IEEE Conference on High Performance Computing, Networking and Storage, Pittsburgh, PA, USA, 2004.
[37] C.D. Guerrero, M.A. Labrador, On the applicability of available bandwidth estimation techniques and tools, Comput. Commun. 33 (1) (2010) 11–22.
[38] D. Katabi, M. Handley, C. Rohrs, Congestion control for high bandwidth-delay product networks, ACM SIGCOMM Comput. Commun. Rev. 32 (4) (2002) 89–102.
[39] K. Ramakrishnan, S. Floyd, D. Black, The Addition of Explicit Congestion Notification (ECN) to IP, RFC 3168, IETF (2001).
[40] B. Briscoe, A. Jacquet, T. Moncaster, A. Smith, Re-ECN: Adding Accountability for Causing Congestion to TCP/IP, Internet Draft, IETF (2014).
[41] B. Briscoe, R. Woundy, A. Cooper, Congestion Exposure (conex) Concepts and Use Cases, RFC 6789, IETF (2012).
[42] 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, Redmond, WA, USA, 2012, pp. 61–66.
[43] J. Postel, Transmission Control Protocol, RFC 793, IETF (1981).
[44] A. Shokrollahi, LDPC Codes: An Introduction, Technical report, Digital Fountain Inc., 2003.
[45] K. Pawlikowski, H.D.J. Jeong, J.S.R. Lee, On credibility of simulation studies of telecommunication networks, IEEE Commun. Mag. 40 (1) (2002) 132–139.
[46] M. Bateman, S. Bhatti, TCP testing: How well does ns2 match reality? in: Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, Australia, 2010, pp. 276–284.
[47] S. Floyd, E. Kohler, Tools for the Evaluation of Simulation and Testbed Scenarios, Technical report, IETF, 2006.
[48] M.P. Fernandez, S. Wahle, T. Magedanz, A new approach to NGN evaluation integrating simulation and testbed methodology, in: Proceedings of the 11th International Conference on Networks, Saint-Gilles, Réunion Island, France, 2012, pp. 22–27.
[49] N. Dukkipati, N. McKeown, Why flow-completion time is the right metric for congestion control, ACM SIGCOMM Comput. Commun. Rev. 36 (1) (2006) 59–62.

[50] S. Molnár, Z. Móczár, Three-dimensional characterization of Internet flows, in: Proceedings of the 46th IEEE International Conference on Communications, Kyoto, Japan, 2011, pp. 1–6.
[51] D.M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, J. Comput. Netw. ISDN Syst. 17 (1) (1989) 1–14.
[52] D.X. Wei, P. Cao, S.H. Low, Time for a TCP Benchmark Suite? Technical Report, California Institute of Technology, Pasadena, CA, USA, 2005.
[53] H. Zhang, Service disciplines for guaranteed performance service in packet-switching networks, Proc. IEEE 83 (10) (1995) 1374–1396.
[54] P. McKenney, Stochastic fairness queueing, Internetworking: Res. Exp. 2 (1991) 113–131.
[55] Z. Móczár, S. Molnár, B. Sonkoly, Multi-platform performance evaluation of digital fountain based transport, in: Proceedings of the Science and Information Conference 2014, London, UK, 2014, pp. 690–697.
[56] Dummynet Network Emulator, http://info.iet.unipi.it/~luigi/dummynet/.
[57] M. Lacage, Experimentation Tools for Networking Research, 2010 (Ph.D. thesis). (available at http://cutebugs.net/files/thesis.pdf).
[58] Network Simulation Cradle, http://www.wand.net.nz/~stj2/nsc/.
[59] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, R. Steinmetz, Modelling the internet delay space based on geographical locations, in: Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, Weimar, Germany, 2009, pp. 301–310.
[60] T.V. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Trans. Netw. 5 (3) (1997) 336–350.
[61] Z. Móczár, S. Molnár, On the Dynamic Behavior of Digital Fountain Based Communication, in: Proceedings of the 58th IEEE Global Communications Conference, San Diego, CA, USA, 2015, pp. 1–6.
[62] G. Appenzeller, I. Keslassy, N. McKeown, Sizing router buffers, ACM SIGCOMM Comput. Commun. Rev. 34 (4) (2004) 281–292.
[63] H. Park, E.F. Burmeister, S. Bjorlin, J.E. Bowers, 40-gb/s optical buffer design and simulation, in: Proceedings of the 4th International Conference on Numerical Simulation of Optoelectronic Devices, Santa Barbara, CA, USA, 2004, pp. 19–20.
[64] S. Molnár, Z. Móczár, B. Sonkoly, How to transfer flows efficiently via the internet? in: Proceedings of the 3rd IEEE International Conference on Computing, Networking and Communications, Honolulu, HI, USA, 2014, pp. 462–466.
[65] HTTP Archive, http://www.httparchive.org/interesting.php.
[66] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H. Madhyastha, Flowsense: monitoring network utilization with zero measurement cost, in: Proceedings of Passive and Active Measurement Conference, vol. 7799, 2013, pp. 31–41.Lecture Notes in Computer Science
[67] S. Chowdhury, M. Bari, R. Ahmed, R. Boutaba, Payless: a low cost network monitoring framework for software defined networks, in: Proceedings of the 14th Network Operations and Management Symposium, Krakow, Poland, 2014, pp. 1–9.
[68] N. van Adrichem, C. Doerr, F. Kuipers, Opennetmon: network monitoring in openflow software-defined networks, in: Proceedings of the 14th Network Operations and Management Symposium, Krakow, Poland, 2014, pp. 1–8.
[69] C.Y. Hong, M. Caesar, P.B. Godfrey, Software defined transport: flexible and deployable flow rate control, in: Proceedings of the Open Networking Summit 2014, Santa Clara, CA, USA, 2014, pp. 1–2.
[70] 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, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, San Jose, CA, USA, 2012, pp. 1–14.
[71] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), ACM SIGCOMM Comput. Commun. Rev. 40 (4) (2010) 63–74.
[72] G. Parisis, T. Moncaster, A. Madhavapeddy, J. Crowcroft, Trevi: watering down storage hotspots with cool fountain codes, in: Proceedings of the 12th ACM Workshop on Hot Topics in Networks, College Park, MD, USA, 2013, pp. 1–6.