MŰEGYETEM 1782

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

Technical Report

Design and Performance Evaluation of the Digital Fountain based Communication Protocol

Authors:
Sándor Molnár, Zoltán Móczár, Balázs Sonkoly,
Szilárd Solymos, Tamás Csicsics

Budapest, Hungary
2012

Contents

1 Introduction

Since the first efficient reaction to the phenomenon of congestion collapse in the early Internet, congestion control, mostly performed by the Transmission Control Protocol (TCP), has played an important role in communication networks [1]. Continuously evolving network environments have made significant research demand on designing more and more effective transport protocols. As a result, several TCP versions have been developed in order to fit the ever-changing requirements of communication networks [2, 3]. Although, current high speed TCP variants provide efficient solutions for many network environments, they all fail to act as a universal mechanism considering heterogeneous and changing network conditions. It seems that there is a little hope that TCP's closed-loop congestion control mechanism could result in such a universal solution in the future.

Another well-known drawback of TCP is its buffer space need, which is at least of root order in the number of competing flows [4]. However, even this requirement imposes a serious challenge in all-optical networks where only very small buffer sizes can be realized due to both economical and technological constraints [5]. On the other hand, optical networks using electro-optic and optic-electro conversion to avoid the previously mentioned constraints, pay the price in high conversion times, which considerably reduces the potential performance of the optical medium.

Concerning the limitations of TCP there is a significant justification to rethink the concept of this transport protocol and design it from scratch, omitting the main TCP-related features, most interestingly its congestion control mechanism. Some ideas have already been proposed and investigated where congestion control was not employed at all. One of these ideas was outlined by GENI (Global Environment for Network Innovations), which advocates a Future Internet without congestion control [6] by suggesting efficient erasure coding schemes to recover lost packets. However, no realization or further refinement of the idea has been published so far with the exception of some related work that we overview in the followings.

A decongestion controller was proposed by Raghavan and Snoeren who studied its benefits [7]. Bonald et al. investigated the network behavior in the absence of congestion control [8]. Their surprising result is the confutation of the common belief that operating a network without congestion control necessarily leads to congestion collapse. López et al. analyzed a fountain based protocol using game theory [9]. They showed that a Nash equilibrium can be achieved, and at this equilibrium, the performance of the network is similar to the performance obtained when all hosts comply with TCP. Botos et al. suggested a transport protocol based on the modification of TCP for high loss rate environment using rateless erasure codes [10]. 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.

In this report we do not advocate any modification to TCP, but rather we consider a clean-slate redesign of the Internet and propose a feasible Future Internet architecture where congestion control is not employed, and end hosts send their data at maximal rates while fair scheduling is responsible for providing fairness among competing flows. Such an architecture can be considered 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. We emphasize the simplicity of our solution since by employing our suggested digital fountain based erasure coding scheme for data transfers, packet loss would become inconsequential, which can considerably simplify network routers and can result in highly reduced buffer sizes, which would favor all-optical networking. Concerning stability, we can also expect improvements since maximal rate sending would result in more predictable traffic patterns by avoiding the high extent of rate variation seen in TCP transmissions. This would make traffic engineering a much easier task as well.

The report is stuctured as follows. Section 2 presents our newly developed transport protocol called Digital Fountain based Communication Protocol (DFCP) including the main design principles and some implementation details. In Section 3 the results of a comprehensive performance analysis carried out in a testbed environment are provided.

2 Protocol Design

2.1 Overview

DFCP is a connection-oriented transport protocol, which can be found in the transport layer of the TCP/IP stack, and similar to TCP it ensures reliable end-to-end communication between hosts. The operation of the protocol consists of three main steps, namely connection establishment, data transfer and connection termination. However, unlike TCP our protocol does not use any congestion control algorithm, just encodes the data using Raptor codes and sends the encoded data towards the receiver at maximal rate making possible to carry out 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 needs an extra overhead, it will be shown in the following section 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 and it has been tested under the Debian Lenny distribution [11].

2.2 Protocol Header

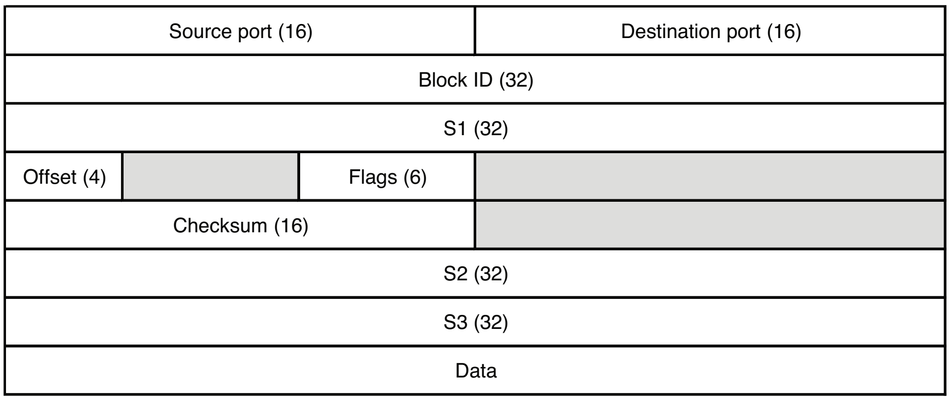


Figure 1. Protocol header structure

The protocol header can be seen in Figure 1 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 to which the given packet belongs. The fields S1, S2 and S3 contain 32-bit unsigned integers, which play roles in the encoding and decoding process. The offset gives the number of 32-bit words in the header, and hence where the first bit of the application data can be found. Flags (e.g. *SYN*, *FIN*) are primarily used in the connection establishment and termination phases, which are discussed in details in the following subsection. The checksum is a generated number depending on the content of the header and partially on the data field.

2.3 Connection Establishment and Termination

DFCP connection establishment is based on a three-way handshake procedure as in the case of TCP [12]. The handshaking mechanism is designed so that the sender can negotiate all the parameters necessary for decoding with the receiver before transferring application data. When the data are successfully received by the destination host, the connection is released similarly to TCP.

2.3.1 Creating a Connection

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

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

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

2.3.2 Closing the Connection

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

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

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

2.4 Coding and Data Transfer

Once a connection is successfully established the protocol is ready to send application layer data. First, data are divided into blocks and each of them is stored in a kernel buffer until free space is available. After that DFCP performs encoding for the waiting blocks sequentially.

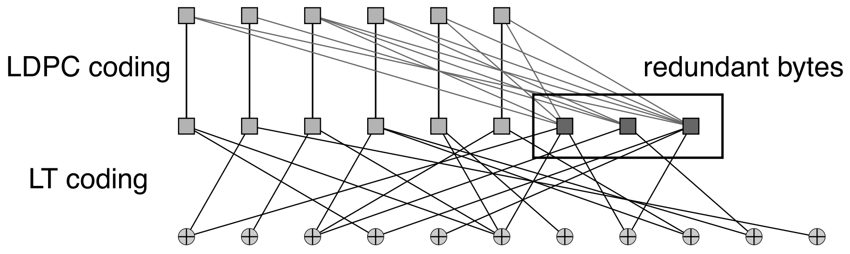


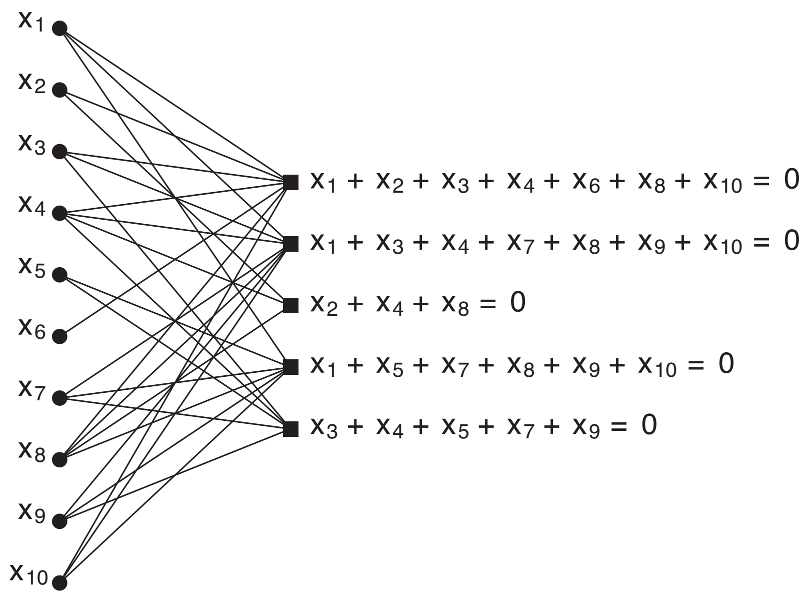
Figure 2. Encoding phases of message blocks

As shown in Figure 2, Raptor coding [13] involves two phases: precoding and LT coding [14]. In our implementation precoding is realized by LDPC (Low-Density Parity-Check) coding [15], which adds some redundant bytes to the original message symbols. LT coder uses the result of the LDPC coding phase as input and produces a potentially infinite stream of encoded bytes.

The concept of LDPC coding is the following. Let us consider a bipartite graph having n_m nodes at the left side and n_c nodes at the right side. The nodes on the left and right sides are referred as *messages nodes* and *check nodes*, respectively. An example is shown in Figure 3. As we can see, for each check node it holds that the sum (XOR) of the adjacent message nodes is 0. 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 0. 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. From the application layer the LDPC encoder receives k bytes and it extends the original message by $n-k$ redundant bytes, and as a result the length of the extended message will be n. In the current implementation of DFCP the size of the original message block is $k = 63536$ and $n - k = 2000$ redundant bytes are added, thus the encoded length is $n = 65536$. It is an important part of the LDPC coding process that a random generator is used at both sender and receiver sides. The intial state of the random generator is determined by three variables (S1, S2 and S3), which are exchanged through the *SYN* and *SYNACK* segments.



x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
$x_1 + x_2 + x_3 + x_4 + x_6 + x_8 + x_{10} = 0$
$x_1 + x_3 + x_4 + x_7 + x_8 + x_9 + x_{10} = 0$
$x_2 + x_4 + x_8 = 0$
$x_1 + x_5 + x_7 + x_8 + x_9 + x_{10} = 0$
$x_3 + x_4 + x_5 + x_7 + x_9 = 0$

Figure 3. Example of an LDPC code

The second phase of the Raptor coding process is the LT coding, for details please see [15]. As mentioned above, messages are divided into blocks, which are considered as units for coding. Each block consists of packets. If a packet is transferred the actual state of the random generator based on the initial state and the block ID is included in its header, and therefore, the receiver can successfully perform LT decoding. The packet header also contains the block ID so that the receiver can determine to which block the given packet belongs.

2.5 Flow Control

To determine if the next block can be sent, a sliding window mechanism is applied at the sender. The window size gives the maximum number of unacknowledged blocks in the network. It can be set to an arbitrary value in DFCP, and the main purpose is to control the burstiness of data transfer. Once a block is received by the destination host, it returns an acknowledgement to the source host. The sliding window shifts by one unit, and the next block is sent. 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. Finally, received blocks can be decoded with high probability.

2.6 Main Parameters

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

- **Window size.** It controls the number of blocks within a window. Since DFCP uses cumulative acknowledgements, they are sent upon the destination host receives the last block of the window, hence, it also determines the frequency of sending acknowledgements.

- **Redundancy.** It gives the number of redundant bytes added to the original message. Knowing the value of this parameter the coding overhead can be exactly calculated, and therefore, the bandwidth waste can be determined. It is crucial for the efficient operation of DFCP how the redundancy paramter is set. The higher the value, the higher the packet loss rate it can resist against. During the performance evaluation we adjusted the redundancy parameter to an optimal value in many cases. Optimal redundancy is the minimum coding overhead assuming a given loss rate that is necessary for successful data transmission and decoding at the receiver side. In the latest version of DFCP the optimal value can be found manually for a given network environment, but it is a possible option to perform this task by an adaptive algorithm of the protocol, so it will be part of our future research.

- **Acknowledgements.** ACKs can be switched ON or OFF that is advantageous for experimental reasons. The purpose of using acknowledgements 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, so it prevents 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.** Encoding and decoding 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. It is very important to separate the coding process from our new mechanism to get a clear picture about its features. Our main purpose is not to optimize the implementation of the Raptor coding scheme because of the fact that now it is an intesively investigated research area. We would like to examine and prepare a novel concept, which may be able to serve as a universal mechanism for Future Internet.

3 Testbed Analysis

To investigate the behavior of the DFCP protocol in realistic network conditions, a comprehensive performance analysis was carried out in a testbed environment for different network topologies and test scenarios. We analyzed the most important performance metrics including goodput, link utilization and flow completion time. To emphasize the beneficial properties of DFCP we compared it to different TCP versions, namely TCP Cubic which is the default congestion control algorithm in the Linux kernel and TCP NewReno with SACK option.

Table 1. Hardware components of test computers

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

(a) Hardware components of senders and receivers

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

(b) Hardware components of the network emulator

The measurement setup consisted of senders, receivers and a Dummynet network emulator, which was used for simulating various network parameters such as queue length, bandwidth, delay and packet loss probability [16]. Each test computer was equipped with the same hardware components according to Table 1.

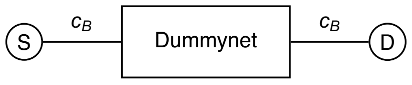


Figure 4. Dumbbell topology with one source-destination pair

The first experiments were performed on a simple dumbbell topology with one source and destination as shown in Figure 4. The measurement duration was 60 seconds for each test, and the flow completion times longer than this duration were calculated by using the steady-state goodput. Regarding the network parameters only the packet loss rate and the round-trip time were varied. The buffer size was set to a high value in order to exclude it from the limiting factors, and the bottleneck link had a capacity $c_B = 1$ Gbps. In these scenarios we used the goodput (i.e. the number of useful bytes transferred per second) as the performance metric.

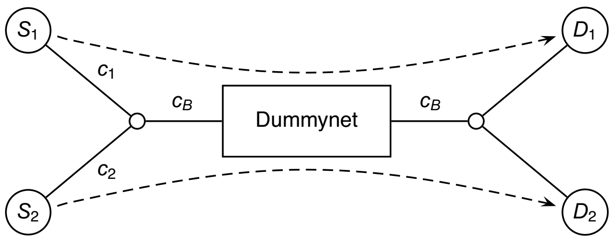


Figure 5. Dumbbell topology with two source-destination pairs

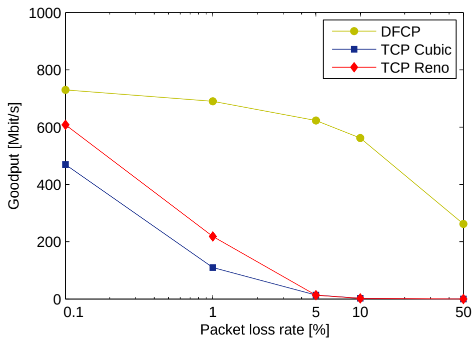
We also performed experiments with two competing flows of the same type to study the fairness properties of the investigated transport protocols. The second measurement setup can be seen in Figure 5 where all parameters were set similarly as described at the first dumbbell topology complemented by the condition $c_1 = c_2 = 1$ Gbps. The flows were started together and we used WFQ (Weighted Fair Queueing) as the scheduling method with equal weights (i.e. 50-50%) [17].

During the testbed measurements the parameters of the Dummynet network emulator were set as below:

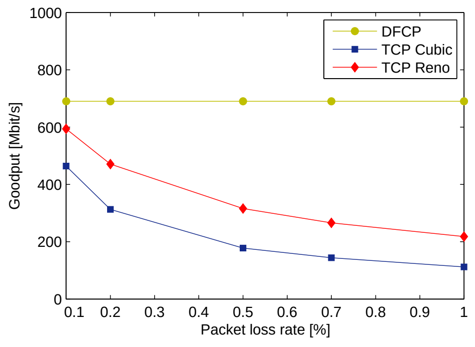
- queue length: $queue = 10000$
- bandwidth: $bw = 0$ (unlimited)
- packet loss rate: plr was changed
- delay: $delay$ was changed

The following subsections present our analysis results. By default we used optimal redundancy parameters in DFCP, but in some cases the redundancy parameter was fixed and adjusted to a certain rate of packet loss that is marked in the caption of the figures.

3.1 Dumbbell Topology with Individual Flows

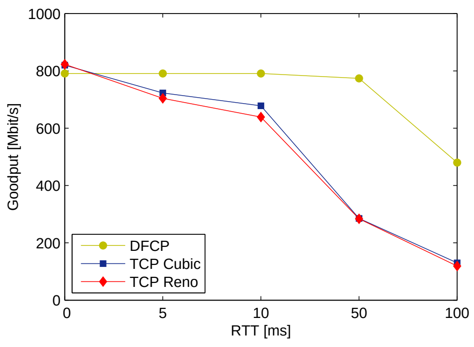


(a) Goodput for increasing packet loss rate using optimal redundancy parameters in DFCP

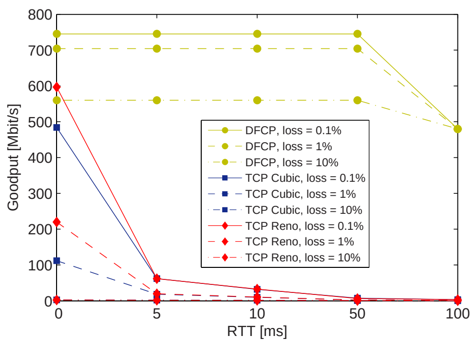


(b) Goodput for increasing packet loss rate using a fixed redundancy parameter in DFCP adjusted to packet loss of 1%

Figure 6. Goodput for increasing packet loss rate with different parameter settings

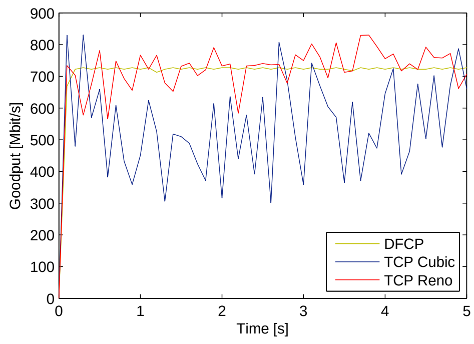


(a) Goodput for increasing round-trip time without packet loss

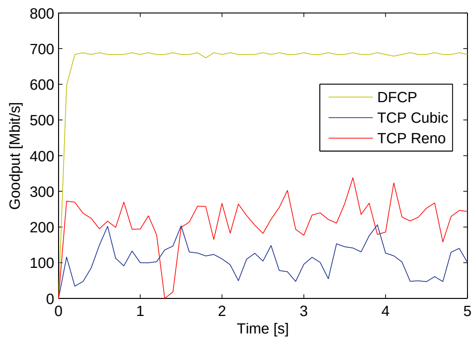


(b) Goodput for increasing round-trip time and different packet loss rates

Figure 7. Goodput for increasing round-trip time

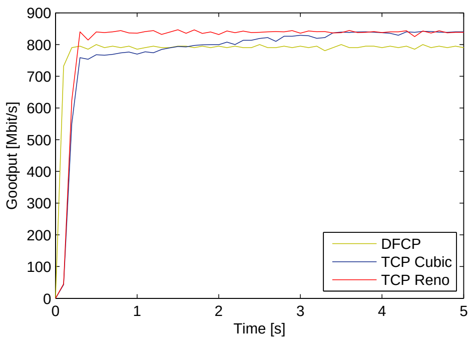


(a) Packet loss rate = 0.1%

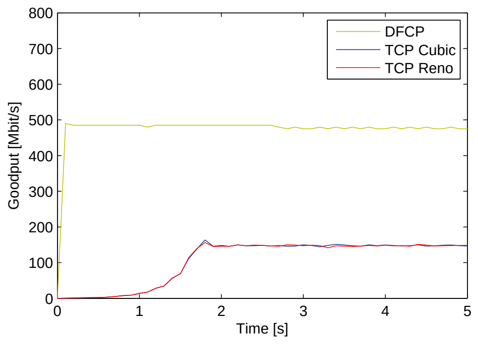


(b) Packet loss rate = 1%

Figure 8. Transient behavior for different loss rates

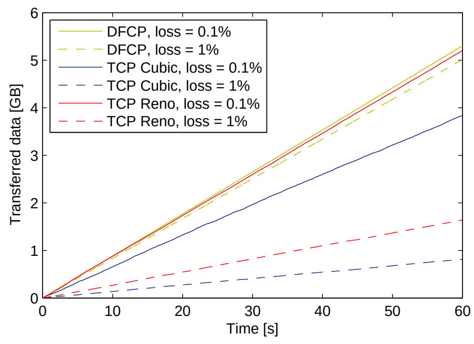


(a) RTT = 10 ms

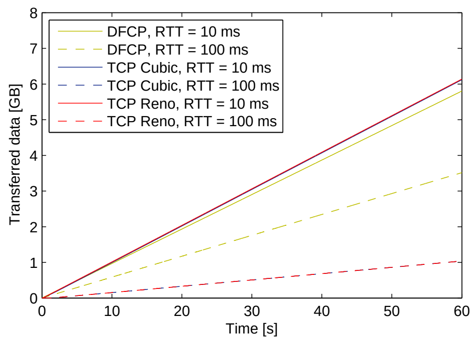


(b) RTT = 100 ms

Figure 9. Transient behavior for different round-trip times

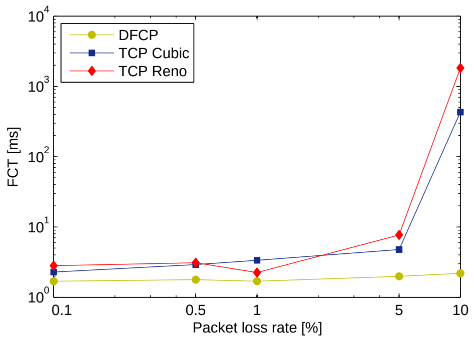


(a) Different packet loss rates

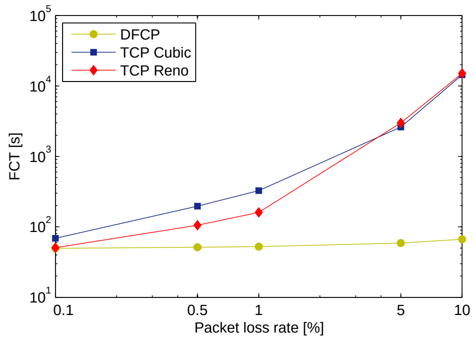


(b) Different round-trip times

Figure 10. Transferred data in function of time for different loss rates and round-trip times

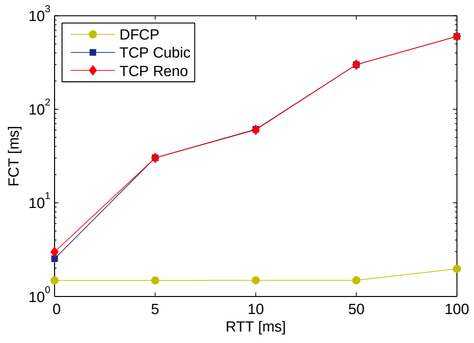


(a) Web object

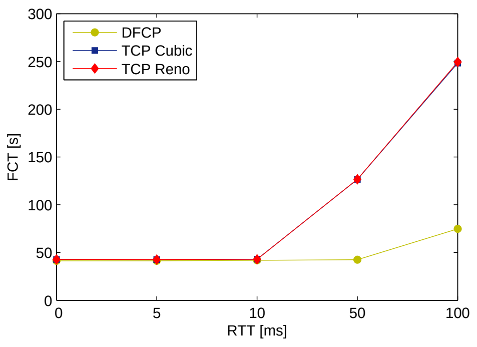


(b) DVD

Figure 11. Flow completion times for different loss rates

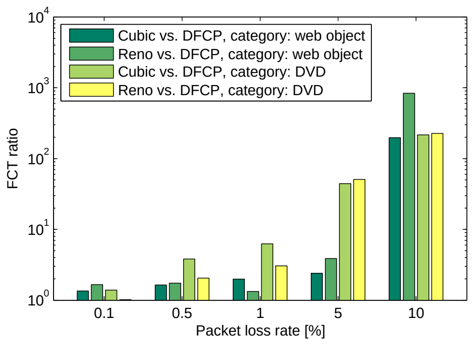


(a) Web object

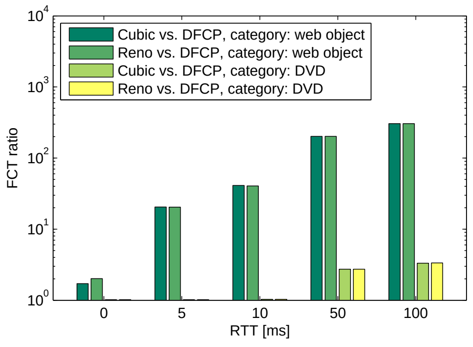


(b) DVD

Figure 12. Flow completion times for different round-trip times

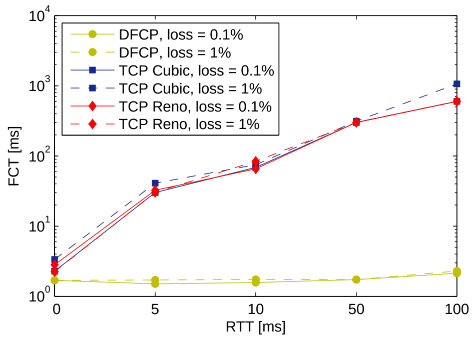


(a) Different packet loss rates

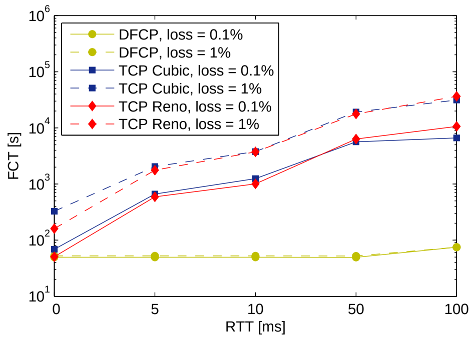


(b) Different round-trip times

Figure 13. FCT ratios for different packet loss rates and round-trip times



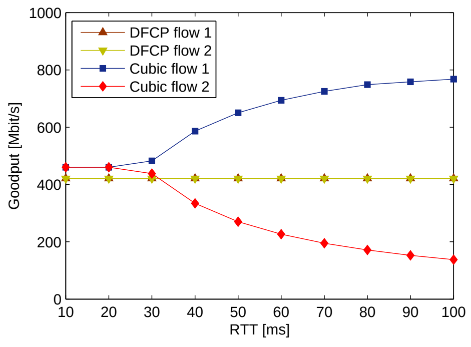
(a) Web object



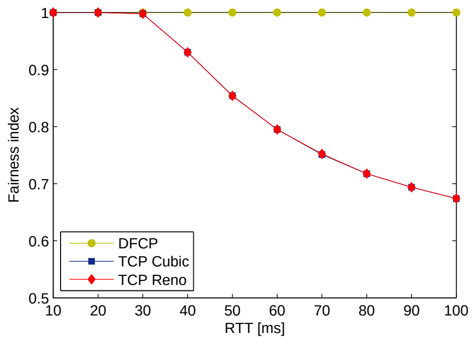
(b) DVD

Figure 14. Flow completion times for different round-trip times and changing loss rates

3.2 Dumbbell Topology with Two Competing Flows

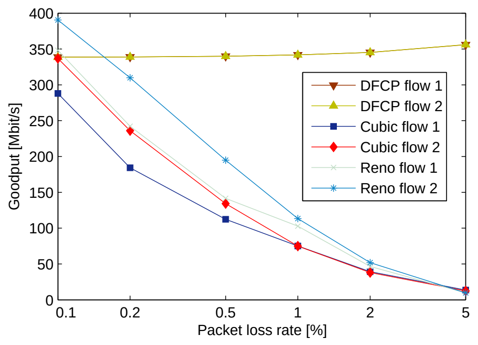


(a) Goodput of DFCP and TCP flows

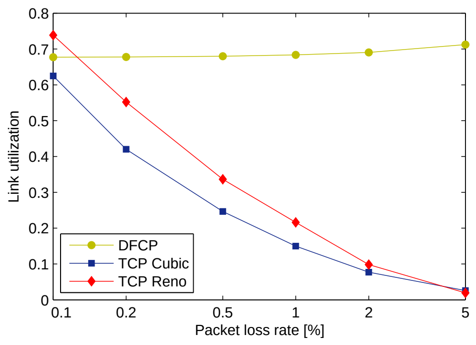


(b) Intra-protocol fairness of DFCP and TCP variants

Figure 15. 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

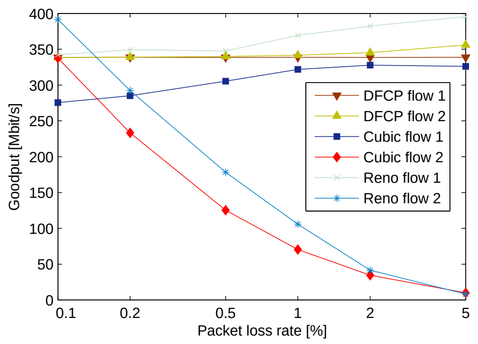


(a) Goodput for increasing packet loss rate

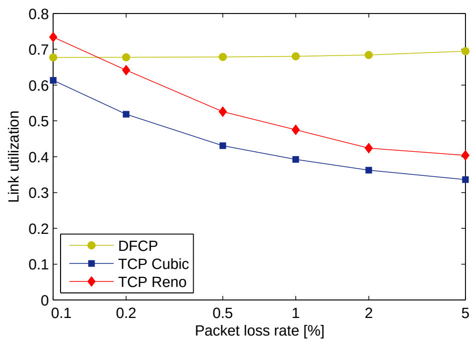


(b) Link utilization for increasing packet loss rate

Figure 16. Goodput and link utilization for two competing flows with equal loss rate

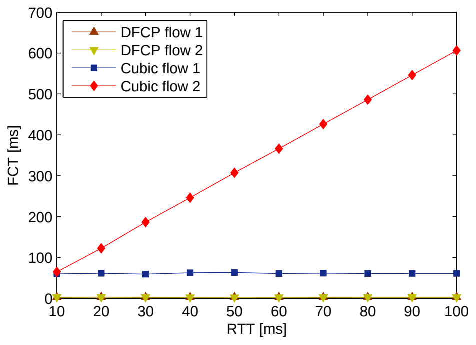


(a) Goodput for increasing packet loss rate

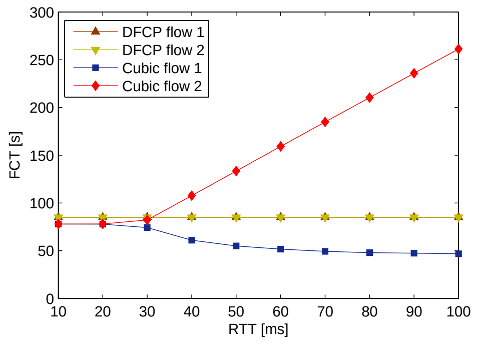


(b) Link utilization for increasing packet loss rate

Figure 17. Goodput and link utilization for two competing flows with different loss rates

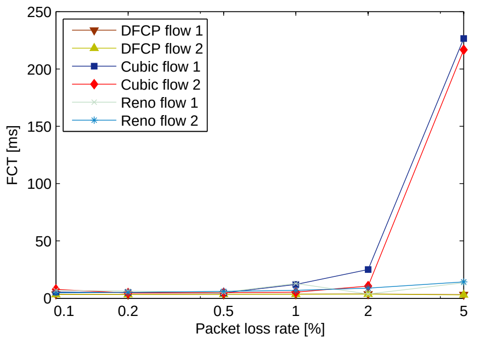


(a) Web object

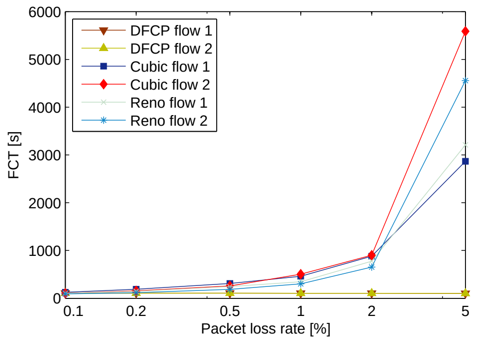


(b) DVD

Figure 18. Flow completion times for two competing flows with different round-trip times, $d_1 = 10$ ms, d_2 is variable

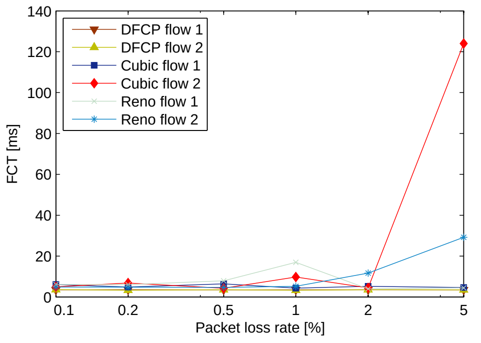


(a) Web object

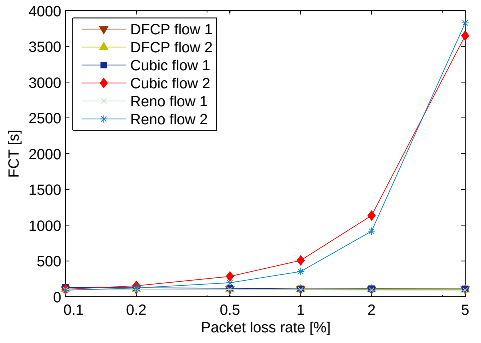


(b) DVD

Figure 19. Flow completion times for two competing flows with equal loss rate (redundancy parameter is adjusted to packet loss of 5%)



(a) Web object



(b) DVD

Figure 20. Flow completion times for two competing flows with different loss rates (redundancy parameter is adjusted to packet loss of 5%), $p_1 = 0.1\%$, p_2 is variable

References

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

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

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

[4] G. Appenzeller, I. Keslassy, N. McKeown, “Sizing Router Buffers”, *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pp. 281–292, New York, NY, USA, 2004.

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

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

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

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

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

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

[11] Sz. Solymos, “Design and Implementation of a Robust Transport Protocol without Congestion Control”, *BSc Thesis (in Hungarian)*, 2011.
`http://hsnlab.tmit.bme.hu/~molnar/files/Solymos_BSc_Thesis.pdf`

[12] J. Postel, “Transmission Control Protocol”, *RFC 793, IETF*, 1981.

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

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

[15] A. Shokrollahi, "LDPC Codes: An Introduction", *Technical Report, Digital Fountain Inc.*, 2003.

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

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