FECTCP for High Error and Large Delay Wireless Networks

Anghel Botoș
Technical University of Cluj-Napoca
26-28 G. Barițiu
Cluj-Napoca, Romania
Anghel.Botos@com.utcluj.ro

Zsolt Polgar
Technical University of Cluj-Napoca
26-28 G. Barițiu
Cluj-Napoca, Romania
Zsolt.Polgar@com.utcluj.ro

ABSTRACT

Current Internet protocols like TCP have been designed to work well on channels with low channel error rates. Wireless channels on the other hand exhibit high loss rates when compared to wired channels. Under these conditions, current TCP implementations mistake losses due to channel errors as being caused by congestion and as a result the sending rate is unnecessarily reduced leading to a degradation in performance. As a result, a discrimination mechanism between channel loss and congestion loss is needed, in order to make TCP perform better over high error rate links. Unfortunately, the implementation of such mechanisms is complicated by the fact that the congestion control and the loss handling mechanisms in TCP are strongly intertwined.

This paper presents an approach for the implementation of a TCP-like transport protocol[1] by using generic rateless erasure correcting codes for the error handling mechanism, thus decoupling it from the congestion control mechanism of the protocol. The congestion control included in the protocol is inspired from current TCP implementations, also containing a discrimination mechanism between channel loss and congestion loss.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Architecture (OSI model)*; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet*

General Terms

Design, Performance

[1]i.e. reliable, connection-oriented and having end-to-end flow control

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM CoNEXT 2009, December 1–4, 2009, Rome, ITALY.
ReArch '09 Rome, Italy
Copyright 2009 ACM X-X-X-X/XX/XX ...$5.00.

Keywords

TCP, rateless erasure codes, congestion control

1. INTRODUCTION

Over the Internet, the physical channel is seen at the higher layers as a packet erasure channel, due to the fact that packets with errors are usually not delivered to the higher layers. Most protocols used nowadays over the Internet have been designed to operate under the low loss rates specific to wired channels. Wireless channels on the other hand exhibit significantly higher packet loss rates.

The *Transport Control Protocol* (TCP)[8] provides reliability over the packet erasure channel by using acknowledgements and retransmission of unacknowledged packets. It also uses a flow control mechanism in order to detect congestion and adjust its sending rate accordingly. This mechanism makes almost no distinction between losses arising from congestion or losses arising from errors on the physical channel, treating all losses as congestion. For low packet loss rates, this mechanism offers sufficiently good performances, proof for this being the success of TCP/IP. Switching over to channels with higher loss rates and treating all packet losses as congestion in the network will lead to a degradation of the performance of TCP. Several analitycal throughput models exist for TCP, and all lead to the conclusion that TCP's performance, in terms of achievable steady-state throughput, degrades significantly in the presence of high packet loss rates [3][7]. Various improvements to TCP have been made with the purpose of distinguishing between losses caused by congestion and losses originating from errors on the transmission channel [4][10]. Still, even with these improvements, TCP does not perform satisfactory on channels with high segment loss rates.

Since the physical channel is seen at the higher layers as a packet erasure channel, this opens the possibility of using modern erasure correcting codes to ensure the reliability at the transport layer. The most appealing codes for this task are rateless erasure codes like Digital Fountain Codes [2] or network coding techniques used for source coding [11]. Current research regard-

ing a new architecture for the Internet is also favoring the integration of coding techniques at various layers in order to improve the performance and the efficiency of the network[6][1].

This paper presents a proposal for the implementation of a TCP-like transport protocol (i.e. reliable, connection-oriented and having end-to-end flow control), that uses generic rateless erasure correcting codes for its error handling mechanism. The purpose of the paper is also to prove that such an approach can lead to a more performant transfer than the regular TCP implementation on channels with high packet loss rates.

The paper is organized as follows: Section 2 presents the general architecture of the protocol in question, Section 3 presents some details regarding the implementation and the scenarios used for the simulations, Section 4 presents the results that were obtained from the simulations and Section 5 presents the conclusions that were drawn from this study.

2. THE PROPOSED PROTOCOL

The protocol proposed in this paper uses rateless erasure correcting codes, like Digital Fountain codes [2] or network coding techniques used for source coding [11], applied to the data that is going to be transferred through the network. The segments sent through the network are symbols from the output of such an encoder. This allows a complete independece between the design of the congestion control mechanism and the error handling mechanism of the protocol. Due to the use of forward error correcting codes the protocol in question will be called FECTCP.

2.1 Rateless Erasure Codes

A general definition for rateless erasure codes is given below:

Definition 1. A *rateless erasure code* is a code which can produce an **almost** infinite set/flow of encoded symbols from any finite set of K input symbols. The decoder of such a code can recover all the K input symbols from any $(1+\varepsilon)K$ encoded symbols with arbitrarily high probability as K grows, for some $\varepsilon \in \mathbb{R}^+$.

2.1.1 Encoding and Decoding

Let $F = (GF(2^m))^n$ be the set of all $m \times n$ bit length strings, for $m, n \in \mathbb{N}^*$. The elements of this set are n-dimensional vectors having their coordinates elements from $GF(2^m)$. Let $\mathcal{F} = (F, \oplus)$ be the finite n-dimensional vector space over F induced by some addition operator $\oplus$. The input symbols for the rateless erasure code and the code-words at the output of the encoder are elements of $\mathcal{F}$. Furthermore, choose $V = GF(2^m)$ as the set of scalars and introduce some multiplication operator, $\otimes$, such that $\mathcal{G} = (\mathcal{F}, V, \otimes)$ forms a vector field over $\mathcal{F}$.

The data that is to be encoded is partitioned into $K \in \mathbb{N}^*$ input symbols, denoted by $\mathbf{x}_i \in \mathcal{F}$, $i \in 1..K$. If the actual length of the data is not an integer K times $m \times n$ bits, the data can be padded with "0" bits up to the next integer value of K. Whenever the encoder needs to produce an output symbol (encoded symbol) $\mathbf{y}_j$ it chooses K random elements from V, a_{ji} according to some distribution. The encoded symbol will be:

$$\mathbf{y}_j = \sum_{i=1}^{K} a_{ji}\mathbf{x}_i \tag{1}$$

where the addition and the multiplication operations are the ones defined by the $\oplus$ and the $\otimes$ operators, respectively.

The decoder at the receiver is in possession of $J \geq K \in \mathbb{N}^*$ encoded symbols and the random coefficients a_{ji} that were used by the encoder when generating these J output symbols[2]. As a result, the decoder can form the following system of equations:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_J \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1K} \\ a_{21} & \dots & a_{2K} \\ \vdots & \ddots & \vdots \\ a_{J1} & \dots & a_{JK} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_K \end{bmatrix} = \mathbf{A} \times \mathbf{X} \tag{2}$$

Recovering the K input symbols amounts to solving the system of equations given in (2) using any algorithm suitable for this task[3]. In order for (2) to be solvable, the matrix $\mathbf{A}$ needs to be of full rank, i.e. K. The number of encoded symbols needed for successful recovery of the input symbols is on average $n = (1+\varepsilon)K$, where ε denotes the average overhead of the code. The probability of successful recovery of all input symbols can be made arbitrarily close to 1 for any given $\varepsilon \in \mathbb{R}^+$.

Due to the rateless property of these codes they are extremely good candidates for offering efficient reliability for a wide interval of loss rates.

2.2 Operation of the Proposed Protocol

The proposed protocol uses the *slow-start* and the *congestion avoidance* algorithms with slight modifications. The protocol relies on a dynamic congestion window of size W^c segments, which is also the sliding transmission window, having the initial size of one segment. All segments are uniquely numbered, in order to keep the synchornization between the random number generators used in the encoders and decoders at both ends of the connection. The segments sent through the network are encoded symbols obtained from the output of some rateless erasure code applied to the input data. As a

[2]Various methods can ensure this, e.g including this information in packet headers, or synchronization of the random number generators, etc.

[3]e.g. Gauss-Jordan elimination, upper triangularization with backsubstitution, etc.

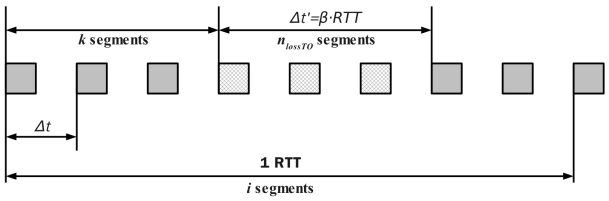

Figure 1: Inter segment distance

result, the terms segment and symbol will be used interchangeably throught the rest of the paper. Each time a segment is sent, a timer for that segment is started. The receiver responds to each received segment with an acknowledgement (ACK) for that particular segment. When an ACK arrives at the sender, the transmission window slides to the right to release as many new segments into the network as the ACK validates. ACKs are cumulative, i.e. the ACK for segment n validates all previous segments. If no ACK is received for a symbol before its timer duration, ΔT_0, expires, then a *time-out* (TO) occurs. If a TO event occurs, then $W^{th} = \max([W^c/2], 2)$ and $W^c = \max([W^c/2], 1)$, where W^{th} is the slow-start threshold. The slow-start threshold is the size of the congestion window for which the protocol switches from slow-start to congestion avoidance.

2.3 Congestion Control Mechanism

For each segment that is sent into the network, the protocol stores its timestamp. When the ACK for that specific segment arrives back at the sender, the protocol will compute the round-trip time for the segment. Using this sample of the round-trip time, the protocol will update the mean RTT and the RTT variance. Then, the *smoothed round-trip time* ($sRTT$) is updated according to the following equation:

$$sRTT_{new} = \alpha \cdot sRTT_{old} + (1 - \alpha)\overline{RTT} \qquad (3)$$

for some $\alpha \in (0, 1)$, a protocol parameter and $\overline{RTT}$ the average RTT. The TO timer duration, denoted by ΔT_{TO} is given by:

$$\Delta T_{TO} = (1 + \beta)(sRTT + \sigma_{RTT}) \qquad (4)$$

where σ_{RTT} is the variance of the mean RTT, and $\beta \in \mathbb{R}^+$ is another protocol parameter.

Segment losses caused by congestion exhibit strong correlation, leading to consecutive segment losses. On the other hand, losses caused by errors on the physical channel are mostly random. The segments sent by the protocol are considered to be evenly spaced in time as shown in Figure 1. Setting the TO timer duration to a value that is longer than one RTT, one "hides" singular segment losses from the congestion control mechanism since this choice allows enough time for the ACK of the segment immediately following the lost segment to return to the sender, and thus cumulatively acknowledging the lost segment as well and cancelling its TO timer. When congestion occurs, segments spanning a longer time-interval will be lost, thus making a TO timer expire and as a result, the protocol will reduce its sending rate to cope with the congestion. There is no need for retransmissions here, since the segments sent into the network are encoded symbols from the output of a rateless erasure code. The protocol will just send the next encoded symbol produced by the encoder, whenever it can send one segment. Once enough symbols have arrived at the receiver, it will be able to recover the initial data, and it will report this back to the sender.

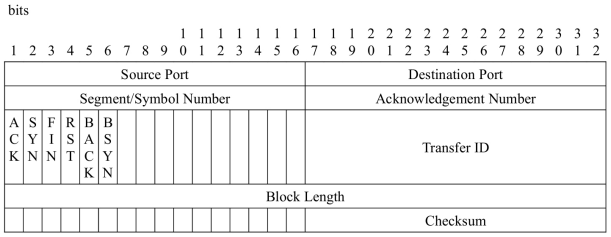

Figure 2: Header structure for the proposed protocol

3. IMPLEMENTATION AND SCENARIO

The presented protocol was implemented using the OMNeT++ network simulation platform together with the INETMANET model library. The header structure for this protocol uses the TCP header as a model and is presented in Figure 2. The header size is 20 bytes, just as the one used by TCP. Some parts of the header are not used, but this is a minimal implementation for this protocol, a real-world implementation might need some additional fields which will hopefully fit in the remaining free space of the header. The meaning of each of the fields within this header is explained in the following.

Source Port and Destination Port.

These fields have the exact same meaning and use as in TCP.

Segment/Symbol Number.

Uniquely identifies a symbol in the flow. The segments are numbered in a continous manner, starting from some value[4] and then wrapping around when the maximum value is reached. Rateless erasure codes have a random structure, each instantiation of such a code resulting in a different structure[5]. The structure of the code needs to be knwon at the decoder as well. Using this field to keep the random number generators

[4]Either a predetermined value, e.g. 0, or a random value.
[5]Although with the same average performances for a given set of parameters

(RNGs) at the encoder and at the decoder in synchronism, the decoder can reconstruct the same code code strucutre as the encoder.

Acknowledgement Number.

This field reports acknowledgments from the receiver back to the sender. The header structure allows for piggybacking of *ACK*s onto data segments in the reverse direction. Currently this is not yet implemented.

Transfer ID.

Since rateless erasure codes are best suited for encoding large amounts of data, this protocol is designed for the same purpose. When an application wants to send some data it only needs to provide the protocol with a "*handle*"[6] to the data. The protocol regards this data as one block and, based on its length it chooses the appropriate parameters for the code that is going to be used. This whole data block is encoded using some rateless erasure code. After the whole block has been successfully transferred, the protocol will report back to the application allowing a new transfer. Several such transfers can occur during the lifetime of a connection. The *Transfer ID* field is used to distinguish between different such transfers and it is also used to perform the initial synchronization of the RNGs during each block transfer. The value of the *Transfer ID* is present in the header of each segment belonging to that data block.

Flags.

The flags used in the header have the following meaning:

- **ACK**, **SYN**, **RST** and **FIN** exactly as in TCP during the connection establishment and teardown;
- **BSYN** and **BACK** are the flags signalling the beginning of a new block of data, respectively the successful recovery by the receiver of the current data block. In case the **BSYN** segment is lost, the receiver can infer the beginning of a new block from the *Transfer ID* and the *Symbol/Segment Number* fields. Once the receiver recovers the content, it will respond to any incoming data segments with a segment containing the **BACK** flag until the sender receives a **BACK** segment.

Block Length.

This field represents the length of the current data block in bytes. This value is present in all segments belonging to the current block. Being represented on 32 bits it allows the protocol to transfer up to 4GB as one block, which is more than sufficient for practical purposes.

[6]Either a file handle or a pointer to the memory area.

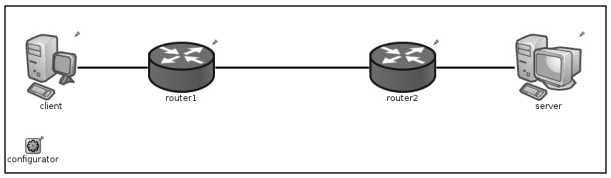

Figure 3: Network scenario using Neyman channel

3.1 Scenario

The scenario used for the comparative study between TCP and FECTCP comprises the two hosts which transfer the data, called *Client* and *Server*, and two routers placed in between and is presented in Figure 3. The access link from the *Server* to its router is Ethernet/802.3 100Mbps. The link between the two routers is a PPP link having a fixed datarate of 1Mbps and introducing a fixed delay during a simulation (thus modelling the Internet cloud between the two routers). No explicit delay jitter modelling is performed on this link, except the delay variations which appear as a result of the enqueueing and dequeueing operations in the routers. The last hop, the link between the *Client* and its router, is a generic CSMA link with a fixed datarate of 1Mbps with no ARQ, which introduces errors on the packets that traverse it. The scenario was chosen for its generality, since this way it is not tied to a specific implementation of the access link of the client, but still allowing for a sufficiently accurate modelling of the entire process. The error channel uses a Neyman Type A distribution for the errors introduced on the data flow [9].

The average overhead of the rateless erasure code in the simulations was set to a conservative value of 5%. Rateless erasure codes with average overhead lower than this value are readily available, for example the Raptor code described in [5] has an average overhead on the order of 0.2%. The β parameter from (4) vas set to a value of 0.3. A preliminary study, not included here, showed that this value ensures good performance of the protocol in terms of throughput, while also maintaining friendliness to other flows sharing the same network resources.

4. EXPERIMENTAL RESULTS

Simulations using the scenario presented in Figure 3 have been run for various values of the end-to-end delay between the *Client* and the *Server* and for various values of the segment loss rate. The block size used for the transfers was 50MB, the results being similar for different block sizes.

Performance depending on end-to-end delay.

The round-trip time range of 10-200ms was studied since these values are typical for connections running

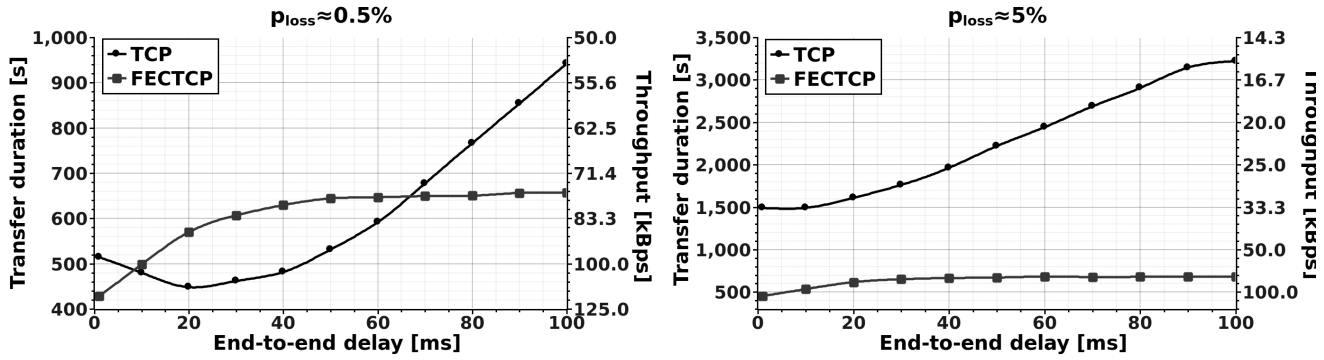

Figure 4: Transfer duration and throughput

through the Internet. Figure 4 presents the results for two different values of the packet loss rate p_{loss}.

As the end-to-end delay increases, TCP exhibits an almost linear increase in the duration of the transfer for both values of the p_{loss}. The cause for this behavior is the limited size of the congestion window for TCP. TCP's behavior for low end-to-end delays is also interesting. For $p_{loss} \approx 5\%$ there is a portion on the graph where TCP has almost constant average throughput. In this case the throughput is limited by the loss rate on the channel. As the end-to-end delay increases, the limitation of the throughput due to the end-to-end delay becomes predominant, leading to the aforementioned increase in the duration of the transfer. This behavior is somewhat present for $p_{loss} \approx 0.5\%$ as well, although at first the transfer duration decreases as the end-to-end delay increases, for low values of the end-to-end delay. The results shows that FECTCP achieves an average throughput that is not degraded by the increase of the end-to-end delay. Although for small end-to-end delays (in the range 1 - 20ms) the performance of the protocol is affected by the increase of the end-to-end delay, it is important to bear in mind that end-to-end delays of this magnitude are rarely seen in practice. Delays usually introduced by the Internet are at least on the order of 50ms. As a result, the situation most often encountered is the one where the average throughput does not depend on these delays. Thus one can conlcude that the proposed protocol can perform better than TCP for high delay networks as well.

Performance depending on segment loss rate.

The range of loss-rates targeted for these simulations was from 0.1% to 25%. Two cases were considered here. The first one shows the behavior of TCP and FECTCP depending, as much as possible, only on the segment loss rate, and not on the end-to-end delay of the link. Therefore, a low value of the end-to-end delay was chosen, in this case 20ms. Figure 5 presents the results of these simulations. The left graph from the figure presents the evolution of the transfer duration and of the average throughput at the application layer as a function of the packet loss rate. The actual throughput that is injected by the transport protocol into the network is presented in the graph on the right. The injected throughput, ρ_{inj} was obtained by adjusting the average throughput at the application layer, ρ_{av}, with the actual segment loss rate. For TCP we have $\rho_{inj}^{TCP} = \frac{\rho_{av}}{(1-p_{loss})}$. For FECTCP an additional adjustment is needed in order to account for the code overhead ε as well. Therefore $\rho_{inj}^{FECTCP} = \frac{\rho_{av}}{(1-p_{loss})}(1+\varepsilon)$.

End-to-end delays as low as 20ms is not realistic. Therefore, the performance of TCP and FECTCP depending on segment loss rate using a real-life value for the end-to-end delay was evaluated. Figure 6 presents the results of these simulations.

These results show that FECTCP performs better than TCP in terms of average throughput once the segment loss rate goes above a certain value (in our case around 2%). For lower segment loss rates TCP performs better, since the performance of TCP is not degraded, while FECTCP needs to account for the code overhead as well. Once the segment loss rate increases, the throughput of TCP will decrease, while FECTCP will maintain its throughput almost the same, and as a result it will be able to successfully transfer the needed data through the network faster than TCP.

5. CONCLUSIONS

The paper proposes the idea of using modern erasure coding techniques, such as rateless erasure codes, integrated into a transport protocol as a mean of improving performance over high loss rate networks. The performance of the proposed protocol, called FECTCP, has been evaluated and compared to the performance of TCP. The dependence of the average throughput as a function of the end-to-end delay and of the packet loss rate was studied. The results showed that FECTCP can indeed perform better than TCP when confronted

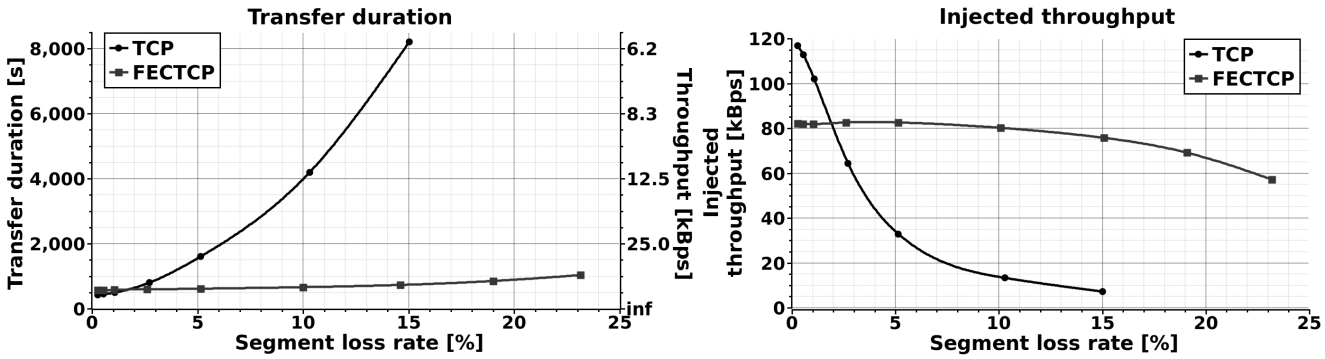

Figure 5: Transfer duration and throughput depending on p_{loss} for end-to-end delay = 20ms

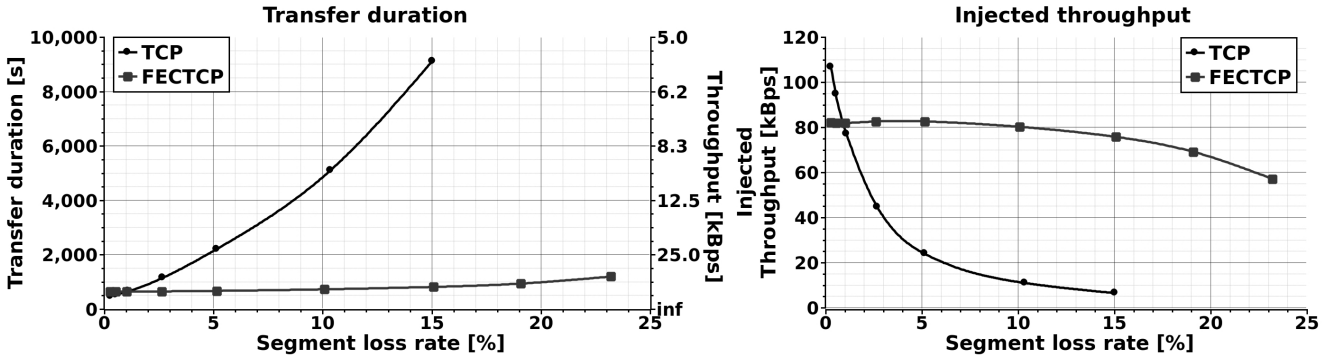

Figure 6: Transfer duration and throughput depending on p_{loss} for end-to-end delay = 50ms

with high paket loss rates and high end-to-end delays on the link. For low packet loss rates (under 2%) and low end-to-end delays TCP still performs better. These results open up additional applications of cross-layering techniques. An example would be to replace the current transport layer with a "transport service". Applications issue requests to this transport service stating what kind of service they require, and based on information from the lower layers (e.g. PHY, MAC, etc.), the most suited protocol is chosen for the the data transport.

6. ACKNOWLEDGMENTS

This work was partially funded by the European Union within the FP7-ICT-2007-1-216041-4WARD project.

The views expressed in this paper are solely those of the authors and do not necessarily represent the views of their employers, the 4WARD project, or the EU Comission

7. REFERENCES

[1] P. A. Aranda, T. Biermann, and D. B. et al. FP7-ICT-2007-1-216041-4WARD/D-5.2.0 - Description of Generic Path Mechanism, 2009.

[2] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data, 1998.

[3] S. Fortin-Parisi and B. Sericola. A Markov Model of TCP Throughput, Goodput and Slow Start. 2004.

[4] V. Jacobson. Modified TCP Congestion Avoidance Algorithm, 1990.

[5] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. RFC5053 - Raptor Forward Error Correction Scheme for Object Delivery, 2007.

[6] D. S. Lun, M. Médard, and M. Effros. On Coding for Reliable Communication over Packet Networks. In *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing, Sept.–Oct. 2004, invited*, 2004.

[7] N. Parvez, A. Mahanti, and C. Williamson. An Analytic Throughput Model for TCP NewReno. 2006.

[8] J. Postel. RFC793 - Transmission Control Protocol, 1981.

[9] D. R. Smith. *Digital Transmission Systems*. 2003.

[10] W. Stevens. RFC2001 - TCP Slow Start, Congestion Avoidance, Fast Retransmit, 1997.

[11] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang. *Network Coding Theory*. 2006.