# FECTCP for High Packet Error Rate Wireless Channels

Anghel Botoș*
Email: Anghel.Botos@com.utcluj.ro

Zsolt Polgar*
Email: Zsolt.Polgar@com.utcluj.ro

Vasile Bota*
Email: Vasile.Bota@com.utcluj.ro

*Faculty of Electronics, Telecommunication and Information Technology
Technical University of Cluj Napoca
400020 Cluj-Napoca, Romania

***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 packet losses due to channel errors as being caused by congestion and as a result they unnecessarily reduce the sending rate leading to a performance degradation. For channels with high packet error rates, a discrimination mechanism between channel loss and congestion loss is needed, in order to make TCP perform better over such links.**

**This paper presents an approach for the implementation of a TCP-like transport protocol[1] by using rateless erasure correcting codes for the error handling mechanism. Based on the distribution of packet losses on the channel, the classical congestion control mechanism of TCP is modified, as to better differentiate between packet loss caused by the wireless link and packet loss caused by congestion.**

## I. Introduction

Over the Internet, the physical channel is perceived 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 on wired channels which exhibit low loss rates. Wireless channels on the other hand exhibit significantly higher packet loss rates.

TCP 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. Using this approach on channels with high loss rates leads to a degradation of the performance of TCP. Analytical throughput models for TCP and practical studies have shown this conclusion to be true [1] [2].

Approaches to this problem are varied in the literature. Some rely on "splitting" the TCP connection by using proxies in order to isolate the lossy link [3] [4], while others rely on the use of the ECN feature of IP [5]. The drawback of the first approach is that it requires special entities in the network to act as proxies, which may not be freely available, while the second requires ECN to be supported in all network entities through which the connection passes.

Since the physical channel is seen at the higher layers as a packet erasure channel, this opens the possibility of using erasure correcting codes to ensure the reliability at the transport layer. Rateless erasure codes [6] [7] [8] [9] or network coding techniques used for source coding [10] are an appealing choice for this task.

This paper presents a simple approach for the implementation of a TCP-like transport protocol with good performance on high loss rate links. No additional network entities, like proxies, are needed, and ECN functionality is also not required for its operation. Reliability is ensured using erasure correcting codes while the congestion control uses the distribution of losses on the physical link to discriminate between loss due to errors or congestion.

## II. Protocol Operation

### A. Connection Establishment and Teardown

Connection establishment and teardown is performed similarly as in TCP using a three-way handshake. Additional parameters needed for the description of the codes used by a specific connection are exchanged by the two ends during the connection establishment phase.

### B. Reliability and Error Handling

The segments sent by this protocol through the network are encoded symbols obtained from the output of a rateless encoder like the ones presented in [6] [7] [8] [9] applied to the original data. Since any such encoded symbol is equally valuable in the recovery of the initial data, this protocol has no need for retransmissions and associated buffers and timers. Whenever the source is able to send a new segment into the network, it will obtain a new encoded symbol from the encoder and send it into the network. Each segment is uniquely numbered[2], this ID allowing the decoder at the destination to

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

[2] Similar to the sequence numbers in TCP

generate the equation which yielded the received symbol. The transmission ends when the destination has received enough encoded symbols to allow for the decoding of the original data. The rateless property of the employed codes ensures an efficient data transfer over a wide range of packet loss rates while at the same time allowing a complete independence of the congestion and flow control from the error handling mechanism of the protocol. Due to the use of forward error correcting codes, the protocol in question will be called FECTCP.

### C. Loss Distribution

In order to be able to discriminate between losses caused by errors on the physical link and losses caused by congestion in the core network it is useful to investigate the distribution of losses produced by a physical channel.

Most transmissions experience errors in bursts or clusters, which invalidates the use of the Poisson error model. To characterize this clustering effect, various models for channels with memory have been developed, among them the *Neyman Type A* distribution [11]. One can view this distribution as a compound Poisson model in which error clusters have a Poisson distribution and errors within a cluster also have a Poisson distribution. This distribution is applicable to physical channels with few error generating mechanism like wired channels, wireless channels affected by slow fading or wireless channels affected by relatively constant interference from other users.

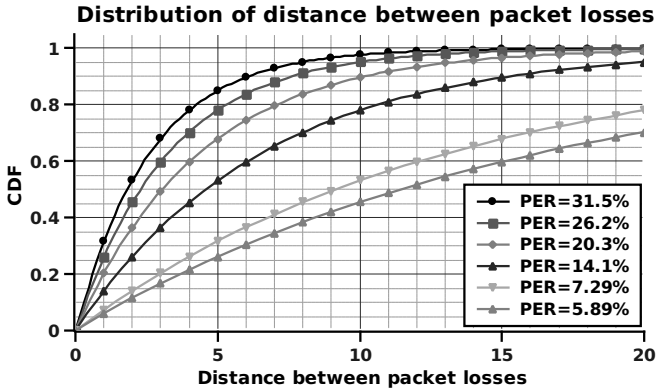


Fig. 1. Distribution of distance between packet losses

The resulting CDF of the distance between two consecutive packet losses is presented in Fig. 1, the packet length being 1500 bytes. From this CDF it is visible that consecutive packet losses occur with a probability that is roughly equal to the average packet loss rate of the link. This observation will be used further on when designing the flow control mechanism of the protocol.

### D. Congestion and Flow Control

For each segment that is sent into the network, the protocol stores its time-stamp. 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:

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

for some $\alpha \in (0,1)$, a protocol parameter and $\overline{RTT}_{new}$ the updated average $RTT$. The protocol also maintains a variable, $cwnd$, the congestion window size, which denotes the maximum number of segments which may be in transit at a given moment. The segments sent into the network and belonging to a given congestion window are evenly spaced in time. This is ensured by the way the protocol schedules segment transmissions. The value of $cwnd$ has no upper bound, as a result there is no limit for the maximum throughput achievable by the protocol even on large delay networks.

The drop-tail queuing policy in routers drops all incoming packets from all flows when the queue is full, thus leading to consecutive packet losses in all flows for the duration of congestion. RED and its variants drop incoming packets with a given probability if the queue size is above a minimum threshold, and then perform like drop-tail once the queue size is above a maximum threshold. This queue management policy, together with the high packet loss rate on the access link, significantly increases the probability of consecutive packet losses to occur within a given flow. This fact can be used as a discriminator between congestion and packet losses caused by errors on the access link.

The flow control used for this protocol is borrowed from TCP, i.e. the slow-start and congestion avoidance algorithm, with some modifications. The idea is to silently ignore singular packet losses, since they are easily attributable to errors on the channel for the high loss rates that we targeted, and to reduce the sending data rate only when consecutive packet losses are encountered in the flow. This is achieved by using cumulative **ACK**s, i.e. each **ACK** acknowledges all previous segments sent into the network[3] and also cancels their timeout timers, and by using a time interval for the timers that is slightly longer than one $RTT$. The amount by which this interval is longer than an $RTT$ is computed in such a way as to allow even the **ACK** for the next segment to arrive back to the sender before the timer expires. Therefore the duration of the timer is given by:

$$\Delta T_{TO} = (1+\beta)(sRTT + \sigma_{RTT}) \qquad (2)$$

where $\sigma_{RTT}$ is the variance of the $RTT$, and $\beta \in \mathbb{R}^+$ is given by:

$$\beta = \frac{1}{cwnd} \qquad (3)$$

This way singular packet losses in the flow are ignored while consecutive packet losses will trigger a reduction of

[3]This does not represent an issue for the reliability of the data transfer since the segments are encoded symbols from the output of a rateless encoder

the sender's data rate. For drop-tail queues and RED queues having the size above the maximum threshold this will have no impact on the correct identification of congestion. For low packet error rates on the wireless link, together with RED queues having the size between the minimum and the maximum threshold packets dropped by the routers will not always necessarily trigger a reduction of the sending data rate.

## III. Implementation and Scenario

The protocol described in Section II was implemented using the *ns2* network simulator. This protocol implementation was used to evaluate its performance compared to other "flavors" of TCP.

The scenario used for performance evaluation is presented in Fig. 2 using one or several WLAN clients performing data transfers using the proposed protocol or various "flavors" of TCP. The WLAN used in the simulations was set at 1Mbps, results for other data rates being similar to the ones presented in Section IV.

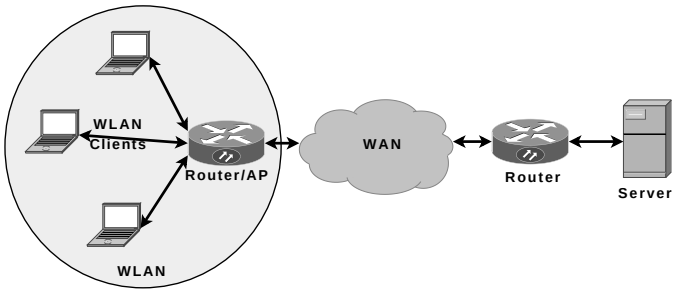


Fig. 2. Scenario used for performance evaluation

Performance was evaluated in terms of average throughput as seen at the application layer[4] and fairness evaluations with respect to other flows like TCP or UDP have also been performed.

## IV. Results

### A. Performance Depending on Segment Loss Rate

The file sizes used for the transfers was chosen to be large enough to allow all protocols to reach the steady-state. File sizes ranged from several MB to tens of MB. Link rates for the links going from the server to the wireless access point have been set to values that allow all simulated flows to be present without the occurrence of congestion. Simulated segment loss rates ranged 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 connection. Therefore, a low value of the end-to-end delay was chosen, in this case 20ms. Fig. 3 presents the results of these simulations.

Simulations have also been performed using more realistic values for the end-to-end delay of the connection. Results for simulations using a 100ms end-to-end delay of the connection

[4]i.e. the time needed to complete a data transfer of a given size.

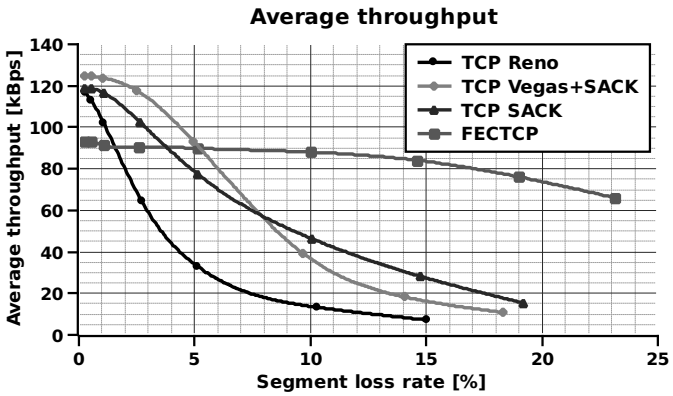


Fig. 3. Average throughput performance for end-to-end delay = 20ms

are presented in Fig. 4. A performance penalty on TCP is visible due to the increased end-to-end delay, but the general behavior depending on the segment loss rate is still the same.

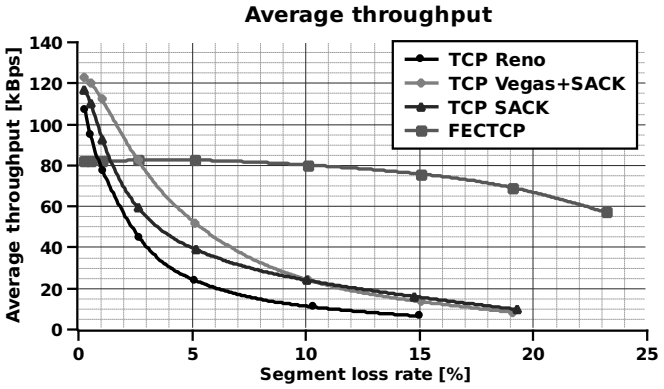


Fig. 4. Average throughput performance for end-to-end delay = 100ms

FECTCP also suffers a small performance penalty due to the increased end-to-end delay, but it still performs better than TCP for high segment loss rates.

Both Fig. 3 and Fig. 4 show that FECTCP can outperform TCP in terms of average throughput for high segment loss rates caused by errors on the physical channel. For low segment loss rates on the other hand (below 5%), it is obvious that TCP outperforms FECTCP. This behavior is easily explained by the fact that the underlying rateless encoding used on the original data requires a certain overhead. This means that the destination needs to receive a number of encoded segments that is slightly larger than the number of original segments. In the simulations a rateless code with an average overhead of 5% was used.

Good performances are obtained in the interval 5% - 25% where consecutive packet losses occur with a low probability due to errors on the channel as was presented in 1. For very large packet loss rates (¿ 25%) the performance of FECTCP starts to decrease as well because the probability of occurrence for consecutive packet losses due to channel errors increases.

### B. Fairness Evaluation and Behavior in the Presence of Congestion

The purpose of this section was to evaluate whether the proposed protocol behaves correctly when faced with congestion on the network, and also to evaluate whether it fairly shares network resources with other competing flows. Several transmissions using FECTCP, TCP and UDP were started on the WLAN clients. The evolution of the instantaneous throughput of all flows was studied.

Fig. 5 presents the evolution of the instantaneous data rate of a TCP flow and a FECTCP flow sharing the same wireless link. The TCP flow starts later than the FECTCP flow. From the figure it is visible that the FECTCP flow allows the TCP connection to have its share of network bandwidth, both of them being able to transfer data at a reasonable data rate.

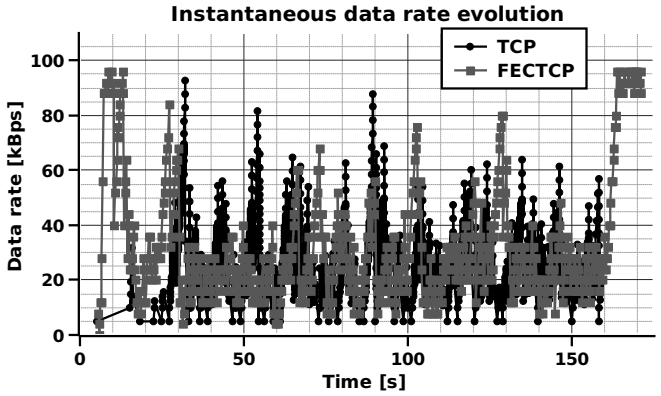


Fig. 5. Evolution of instantaneous data rate of TCP and FECTCP using the same link

Fig. 6 shows the evolution of the instantaneous data rate for a UDP flow and a FECTCP flow using the same wireless link. The UDP flow is a CBR one with a data rate of 20kBps. From the figure one can conclude that FECTCP reduces its data rate when faced with the presence of a UDP flow just as it is expected to do.

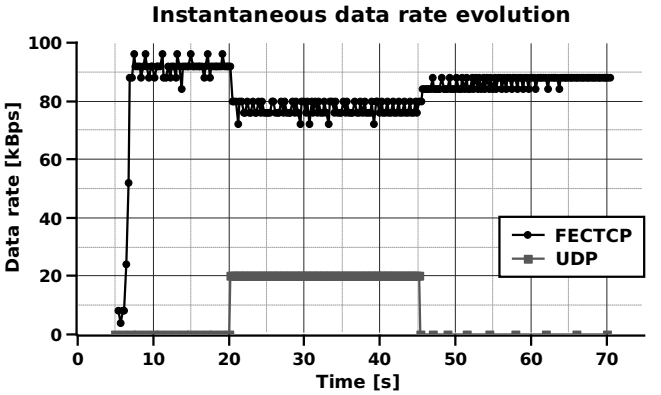


Fig. 6. Evolution of instantaneous data rate of UDP and FECTCP using the same link

Several other simulations have been performed in order to evaluate whether FECTCP behaves in a network friendly manner[5]. All of our simulations show that this protocol behaves in a network friendly manner by fulfilling the above requirements.

## V. Conclusions and Future Work

The paper presented a TCP-like protocol using a congestion control mechanism which takes into account the packet loss distribution on the physical channel. Furthermore, data transfer reliability was ensured by the use of rateless erasure correcting codes, thus eliminating the need for retransmissions and making the implementation simpler. The results obtained through this study show that this protocol can obtain better performance, in terms of average throughput, than TCP on channels with high packet loss rate. Unfortunately this is not a "one size fits all" solution since it performs poorer than regular TCP for low packet loss rates due to the code overhead. Additional evaluation of the protocol also showed that it correctly reduces its sending rate in the presence of congestion and that it fairly shares network bandwidth with other flows using the same physical links. The advantage of the proposed solution lies in the fact that no additional entities, like proxies, and no support for ECN from the network are needed, all flow control operations being performed only in the hosts located at the two ends of the connection.

## Acknowledgment

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

## References

[1] S. Fortin-Parisi and B. Sericola, "A Markov model of TCP throughput, goodput and slow start," *Performance Evaluation*, vol. 58, no. 2-3, pp. 89 – 108, 2004.

[2] N. Parvez, A. Mahanti, and C. Williamson, "An Analytic Throughput Model for TCP NewReno," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, p. 1, November 2009.

[3] A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pp. 136–143, 1995.

[4] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi, "Split TCP for mobile ad hoc networks," *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 1, pp. 138–142 vol.1, 2002.

[5] R. Krishnan, J. P. Sterbenz, W. M. Eddy, C. Partridge, and M. Allman, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks," *Computer Networks*, vol. 46, no. 3, pp. 343 – 362, 2004.

[6] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 56–67, October 1998.

[7] M. Luby, "LT codes," *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pp. 271–280, 2002.

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

[9] P. Maymounkov and D. Mazieres, "Rateless Codes and Big Downloads," in *In IPTPS'03*, 2003.

[10] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, "Network coding theory: single sources," *Commun. Inf. Theory*, vol. 2, no. 4, pp. 241–329, 2005.

[11] D. R. Smith, *Digital Transmission Systems*, 3rd ed. Springer, 2004.

[5] i.e. whether it allows other TCP flows to have a fair share of the existing network bandwidth and whether it yields part of its used bandwidth in favor of UDP when such flows are present