Transport Layer End-to-End Error Correcting

Tal Anker, Reuven Cohen and Danny Dolev

***Abstract*— In this paper we study the advantages and disadvantages of incorporating error correction schemes inside the TCP transport layer protocol. We show that under some circumstances, this scheme achieves better performance than that of retransmission schemes, and, in particular, the TCP-SACK scheme. We analyze the performance of both schemes and discuss the circumstances under which the error correction scheme is favorable.**

I. Introduction

Several suggestions have been raised for the inclusion of error correcting codes in the transport layer [1]. These suggestions have focused on the utility of this approach mainly in multicasting and broadcasting. Here, we wish to establish the idea that the changes in communication and computer hardware make it very beneficial to include an error correcting scheme in the transport layer.

Most computers today are capable of executing about 10^9 instructions per second, and adding extra processing power is cheap relative to the total cost of a web server. This processing power enables execution of several instructions per communicated byte, without a noticeable effect on the CPU load. A simple error correcting scheme, such as adding a XOR of a block of packets as an extra packet to the block, can easily be applied to the setup of most modern computers' CPU, without affecting memory limitations. Furthermore, for static data, such as static web pages, the error correcting data can be prepared and stored in advance, thus requiring no extra processing while sending the information.

The use of error correction in the transport layer has several advantages over other approaches. It allows the inclusion of error correction without rewriting applications. It requires only the end stations to handle the error correction code, adding no additional processing to routers or switches. Also, it allows gradual incorporation into the network, by adding a possible negotiation option to the TCP connection and requiring no changes in network elements along the communication path between the two endpoints. The transparency of this change to other communication layers allows efficient and easy coding, and enables the implementation in either software or hardware with a simple design.

Several advantages exist for the use of error correction over the approach of re-sending lost packets (most notably implemented by the TCP-SACK protocol). The re-sending approach requires a lower quantity of data to be sent, since it induces only the necessary redundancy. However, a single packet loss blocks the window until it is recovered and acknowledged. This limits the size of the window in case errors are present, whereas if error correcting is utilized most errors can be corrected without requiring re-sending. Furthermore, for high delay connections, e.g., an intercontinental cable or a satellite connection, the round trip time becomes very high, leading to long time-outs while waiting for acknowledgments[1]. This, in turn, is pronounced in both long delays in the sending of files and in nonuniform performance and delay, most notably inappropriate for multimedia communication.

In this paper we study the implications of implementing forward error correcting protocols in the transport layer. We use both simulations and analysis to present the benefits of using such an approach.

II. Overview

Many different types of error correcting codes have been studied in the literature (see e.g. [2]). In the transport layer it is usually safe to assume that packets arriving at the destination are free of errors, and that correcting errors inside each packet should be restricted to link-level protocols, since reliability is highly dependent on the link type. The errors we aim to deal with are lost packets. This makes the task of error correcting somewhat easier since the problem of bit flipping is absent, and detecting a lost packet is relatively easy. Thus, one needs to deal only with the reconstruction of lost packets.

Tal Anker is with Radlan inc. and the School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel. Email: tala@radlan.com

Reuven Cohen is with the Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Email: cohenr@shoshi.ph.biu.ac.il

Danny Dolev is with the School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel. Email: dolev@cs.huji.ac.il

[1]This is relevant even in environments where a link-layer FEC is used on the intercontinental link, as the loss can be on another link along the path.

Error correcting schemes for missing packets can be classified as (N, M) schemes, where N is the number of data packets in a block, and M is the number of error correcting packets added to each block. The simplest scheme is the $(N, 1)$ scheme, where each N packet data block is supplemented with one packet containing the XOR of the N data packets. A single missing data packet is easily reconstructed by XORing the other N packets in the block. The operation requires approximately one CPU cycle per transmitted word, and thus induces a low overhead, and is also easy to implement in hardware.

In general, an (N, M) scheme requires any group of N packets out of the total of $N + M$ submitted in each block to reach the destination for the entire block to be reconstructed. The rate of an (N, M) scheme, *i.e.*, the fraction of data packets to total submitted packets, is $\frac{N}{N+M}$. The rate achieves its maximum when a number of error correcting packets is added to a large of data packets.

A. Related Work

This paper deals with a combination of two broad research topics: Forward error correction and the TCP protocol. Obviously, these topics are too broad to be fully covered in this section. Thus, we refer the reader to some major relevant papers and standardization efforts in these areas.

Forward error correction deployed in the networking area has gained much attention in the past decade or so. There are many good references on this area, for instance [3], [4], [5], [1], [6], [7], [8]. General information on error correcting can be found in [2]. FEC in the link layer has already proved itself to be valuable and is standardized and used in the industry. The work in this paper mainly focuses on the usage of end-to-end FEC in the transport layer, and therefore we do not cover link layer FEC in this section. For discussion of modern Tornado codes we refer the reader to [9], [10].

This paper deals with recovering from packet losses done at the transport layer. There are several papers on modelling the Internet behavior, and specifically the packet loss model in the Internet. To understand the Internet behavior in terms of packet loss we refer the reader to [11].

TCP is the most widely used and deployed transport layer in the Internet. Numerous papers and books were written on this subject. The interactions between the TCP congestion avoidance mechanism, flow control, recovering from packet losses and various TCP flavors, are covered in the following papers and standards [12], [13], [14], [15], [16], [17], [18].

Our analysis relies on [19], for the analytical study of TCP throughput and performance.

The work in [1], [20], [21] points out the benefits of using FEC in multicast transport layer. The are also suggestions for incorporating FEC into the application layer [22]. Today, there are several standards developed by the IETF that incorporated FEC in the reliable multicast transport layer [23], [24].

III. Implementation

A. Placement of the Error Correcting Scheme

The most natural place to incorporate packet level error correcting is in the transport layer. The physical and the datalink layers may incorporate some bit level error correcting codes to overcome a single hop reliability. End-to-end error correction should be addressed at the transport layer, since it is responsible for packet reliability. Adding it to a layer that is not targeted to deal with reliability produces a less efficient implementation. For example, TCP deals with end-to-end reliability, and incorporates retransmitting mechanism with congestion avoidance. If FEC were implemented at the application layer, any single packet loss could result in throughput reduction and/or retransmission, although the application layer would be able to recover from such loss. The only way to avoid this is to mix the application and TCP layer, which is both messy and complicated. In addition, updating only the TCP layer will allow a new standard, incorporated directly into the operating system such that existing applications can use this feature in a transparent manner.

In most FEC implementations in the transport layer, FEC is used in conjunction with UDP or a proprietary protocol. Using FEC in conjunction with UDP is desirable since no retransmission mechanism exists in UDP, unlike TCP. Furthermore, in applications like multicast, where each client may miss different packets, retransmission is highly inefficient and FEC is necessary. However, in many cases reliable transmission is required. Only the TCP retransmission mechanism allows reliable transmission even in the face of congestion and multiple losses uncorrectable by FEC. As we show below using FEC with TCP is advantageous in some cases in terms of performance. We therefore propose the usage of the combination of mechanisms: FEC and ARQ.

B. Placement of the FEC Sublayer

There are two possibilities of incorporating error correcting schemes into the transport layer; either TCP that wraps FEC, or FEC that wraps the TCP.

In the first case, the error correcting scheme below the regular TCP algorithm is done by sending the regular TCP output to the new FEC sublayer instead of the IP layer. The sender's FEC sublayer then expands the output by adding M checksum packets to each N data packets and submits the data again to the TCP for the second and final processing. This time the TCP layer passes the packets to the IP layer for transmission. The receiver's FEC sublayer then receives the packets and tries to extract the original data from the received packets. If the data can be extracted, it is delivered to the TCP layer, which is oblivious to any missing packets that had been corrected. If the original data cannot be restored, the receiver behaves as in a regular packet loss event. At this point, the regular TCP mechanism will try to recover the encoded lost packet. Once enough packets are recovered, the original message can be constructed.

An alternative approach is wrapping the TCP with the FEC sublayer. In this case the data sent by the application is expanded by the FEC sublayer to also incorporate error correcting packets. The data is then numbered and submitted to the TCP layer. On the receiving side, the FEC sublayer identifies whether enough packets have arrived. If enough packets have arrived, they are forwarded to the TCP, with an indication to ignore any potentially missing packets. The upper part of the FEC sublayer reconstructs the original message and forwards it to the application. If not enough packets arrive the packets are forwarded to the TCP layer to handle the packets' loss.

In both cases no perfect separation of the FEC sublayer from the TCP layer is possible, and the TCP layer itself must be changed. This is due to the need to change packet sequence numbers, timeouts and considerations of window size.

Each of the approaches has its own advantages. The first approach does not require any change in the interface to the upper layer, though it significantly complicates the implementation. The second approach, on the other hand, is much simpler to implement, since the transmitting part remains the same, though the Receiving part requires some changes. The usual TCP's sequence numbering can be used without the TCP layer ever knowing that the actual transmitted data is not the original data handed of the application. Furthermore, since in the first approach, the TCP layer is responsible for splitting the data into packets, the FEC layer should wait for N packets to arrive to add the error correcting code. A timeout should therefore be added, in case only a small amount of data is sent.

C. Header Changes

Since the receiver should be able to decode the data and reconstruct the original message, both of the approaches above require adding information regarding the location of each packet in the block and the total block size. This allows the sender to change the block size at will, adapting to network conditions, and also on the status, e.g. sending fewer error correcting packets at the window end.

D. Window Size Handling

Since up to M errors in a single block can be corrected using the FEC, it is most appropriate to ignore those correctable errors and to continue the window size update as if no errors occurred. This is the most reasonable decision, since no retransmission is needed in such cases, and therefore the normal data flow is not affected by those errors. An advantage gained by this approach is that a single error does not decrease the window size dramatically, as in regular TCP. Thus, this approach also results in a more well behaved and less fluctuating data rate, and possibly leads to a more predictable, and more efficient network performance.

E. TCP Friendliness

The suggested approach is TCP friendly. It adds a feature to TCP that can be turned on/off at will, depending on the error rate and other channel properties. A single negotiation stage is needed at the connection establishment, and from that point on this feature can be activated at the sender's will. Several levels of error correction can be implemented and switching between them can be done at the sender's will, without a need for any further negotiation. The addition to the header is sufficient for the receiver to determine the error correcting level and the decoding required.

The main issue in considering TCP freindliness is the behavior in face of congestion. It is expected that adding a mechanism which decreases the window size less frequently will result in less effective congection handling. However, since in case of congestion a significant part of a transmitted window is expected to be lost, the decrease in the window size is expected to appear in this case. The window size will not decrease in the face of sporadic losses, where it is not desired. This should actually result in improved throuput and predictablity of the traffic through each node. In case several flows are present the TCP-FEC flow will not starve other flows, since its response to (real) congestion will be similar to that of regular TCP. Furthermore, as can be seen, its overhead over regular TCP in terms of total traffic is

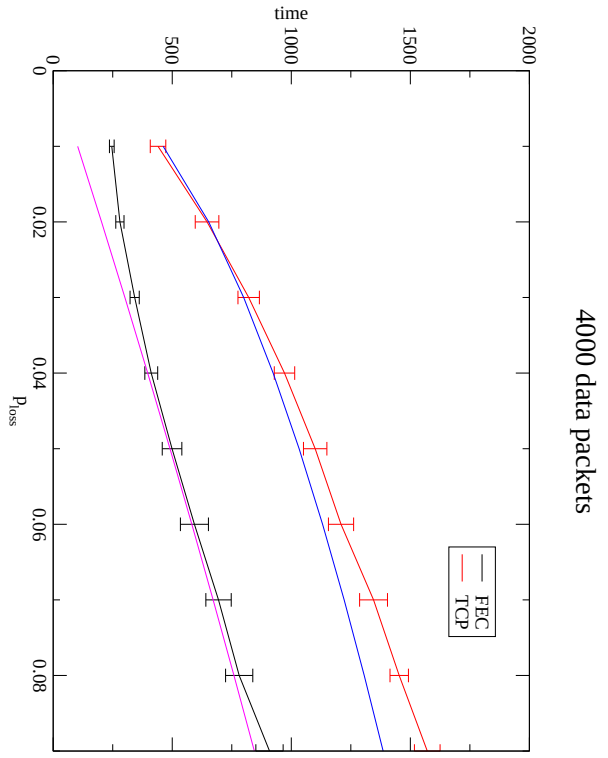

Fig. 1. Long-Lived Session (4000 data packets). Error bars represent standard deviation (here and throughout).

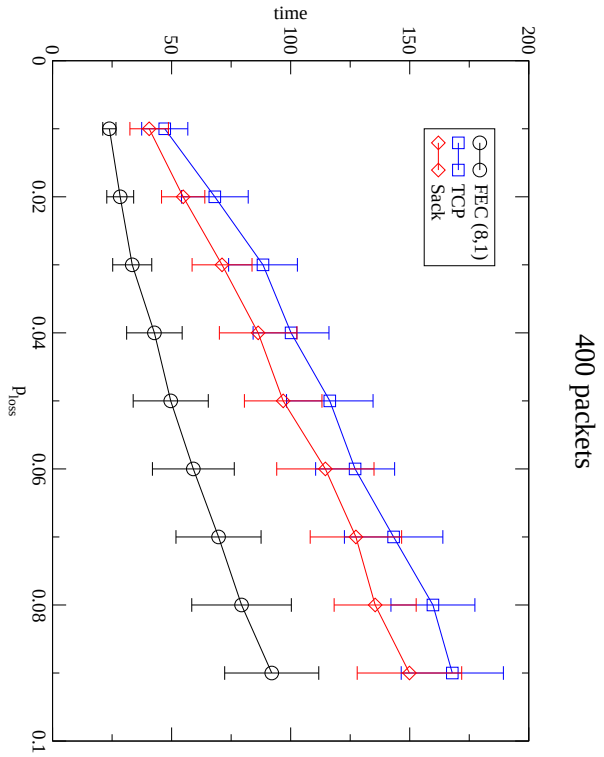

Fig. 2. Medium Size Files (400 data packets).

marginal, while its utilization of existing bandwidth is more efficient.

The retransmission mechanism used is independent of the FEC implementation and either TAHOE, RENO, SACK or any other similar mechanism can be used. The only mechanism affected by the FEC is RED, since the window size is not decreased as a result of a single error[2]. Therefore, the TCP-FEC combination can be considered TCP friendly.

IV. Simulation Results

To validate the performance of the TCP combined with FEC capabilities, we have performed various simulations, using the NS-2 [25] network simulator. We present a comparison of TCP-FEC protocol compared to TCP (TCP-TAHOE) and TCP-SACK.

We have conducted several types of simulations using 1500 bytes packets: One is for "large" file transfer, in which the files are big enough to accommodate 4000 packets. The second is for medium size objects, which are translated into 400 packets per TCP session. The third was for short objects, of 80 packets each ($\approx 120KB$). We also conducted simulations for the special case of 8 packets session, which correspond to about 12KB per session (for small objects such as images contained in an HTML page).

Figure 1 presents the results of simulations of 4000 packets session, with FEC parameters of $N = 8$, and $M = 1$. In the next section we present the analysis for this specific case. The figure presents the total transmission time as a function of the loss probability. We have compared the performance of TCP-FEC with TCP-TAHOE and TCP-SACK. As the graph shows, TCP-FEC is usually more than twice better than regular TCP and TCP-SACK. If the probability of a single packet loss is high $(p > 0.06)$, then the performance is tripled. However, when the error rate further increases, the probability of two packet loss within a stream of $M + N$ increases, and thus the benefit of TCP-FEC decreases. Naturally, in such cases, it would be better to use $M = 2$ (as can be seen in Figure 3).

Figure 2 presents similar results for 400 packets session, with FEC parameters of $N = 16$, and $M = 2$.

In Figure 3 we compare the TCP-FEC using several block parameters with different overhead values. As one

[2]One can think of a mechanism to incorporate both.

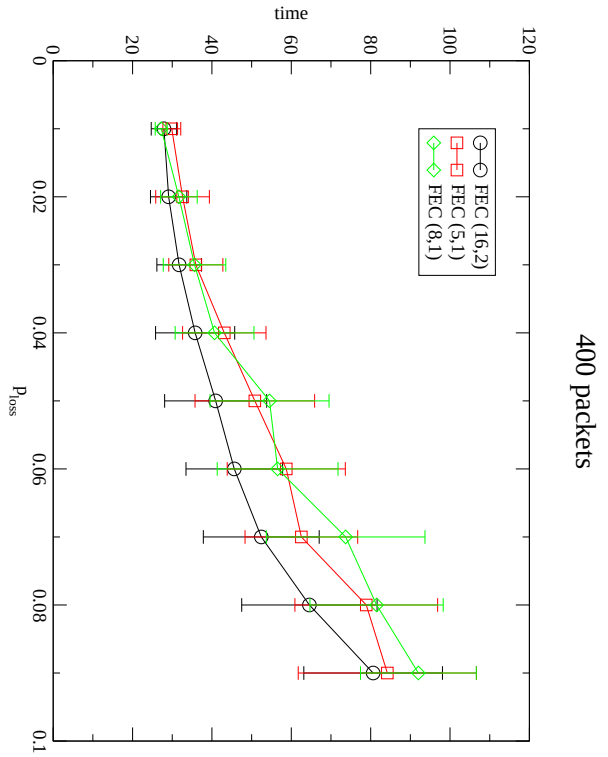

Fig. 3. Different Block Parameters.

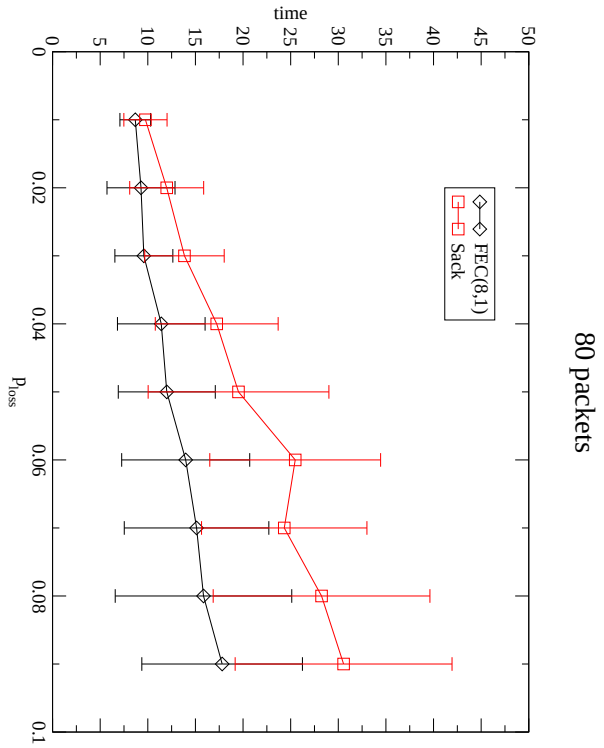

Fig. 4. Short Session (80 data packets).

can see, the $N = 16, M = 2$ parameter set yields the best results. The possibility to sustain 2 packets loss per block is enough for quite high loss probabilities, thus yielding very good results. As for the cases of $N = 8, M = 1$ and $N = 5, M = 1$, for low loss probability, the low overhead of the $N = 8, M = 1$ case yields better results than those of the $N = 5, M = 1$ case. However, as the loss probability increases, the larger overhead of $N = 5, M = 1$ yields results in a better performance.

Figure 4 presents the result of a simulation of 80 packets session, with FEC parameters of $N = 8$, and $M = 1$. Here we compared the performance of TCP-FEC with TCP-SACK. As the graph shows, TCP-FEC is usually more than twice better than TCP-SACK also for this range of file size.

To simulate very short files we studied the case of files of size 12KB (we used 8 packets per file) and assumed 5% packet loss. We found that FEC with $(N, M) = (8, 1)$ required on average about 90% of the running time of TCP. The standard deviation of the running times of FEC was also lower than that of TCP.

To study the overhead of using FEC we conducted a simulation in which we compared the total number of packets, including ACKs of 400 data packets session, with FEC parameters of $N = 8$, and $M = 1$. Figure 5 presents the ratio of the total number of submitted packets in FEC over the number in TCP-TAHOE. The results show that the overhead is only about 5–12%.

V. Analysis

To analyze the performance of TCP-FEC we first analyze the performance of TCP. We follow closely the analysis in [19].

Recall the definition of loss event: a packet is lost, and the receiver replies with a triple-ACK, which is the loss event indication. The effective bandwidth of TCP, given that the probability of a packet loss event is p, is given by,

$$B(p) = \frac{E[Y]}{E[A]} \; . \tag{1}$$

where Y_i is the number of packets sent in the period between the i'th and $i + 1$'st loss event indications, and A_i is the duration between these indications. Now suppose that α_i is the first lost packet since the last loss indication, and that W_i is the window size at that point. Until notification of the loss event arrives (using a triple-ACK), another full window can be sent. Therefore, the

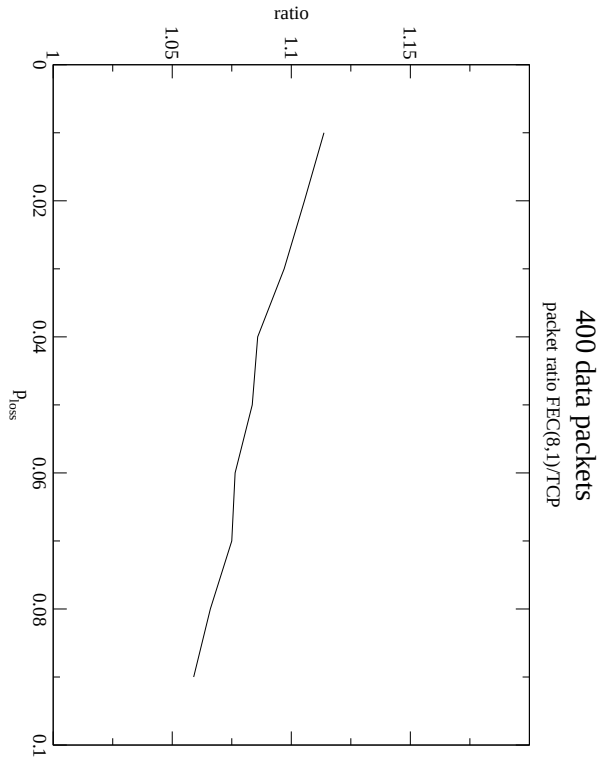

Fig. 5. TCP-FEC vs. TCP-Tahoe Overhead.

number of packets that will arrive at the destination will be $\alpha_i + W_i - 1$, and the expected value is,

$$E[Y] = E[\alpha] + E[W] - 1 \ . \tag{2}$$

The first loss event is distributed according to the geometric distribution. Thus,

$$P[\alpha = k] = (1-p)^{k-1}p, \qquad k = 1, 2, \cdots \ , \tag{3}$$

and the expected value is

$$E[\alpha] = \sum_{k=1}^{\infty}(1-p)^{k-1}pk = \frac{1}{p} \ . \tag{4}$$

This leads to

$$E[Y] = \frac{1-p}{p} + E[W] \ . \tag{5}$$

Denote the round trip time for the j'th window after the i'th loss indication as r_{ij}. The r_{ij} values can be considered identically distributed and independent. If the loss α_i appeared during the X_i window after the previous loss indication, the expected total time between loss indications is

$$E[A] = (E[X] + 1)E[r] \ . \tag{6}$$

Let us denote $E[r] = RTT$.

Recall that TCP reacts to losses by reducing the window size to half, i.e., if the window size at the detection of the i'th loss was W_i, it is reduced to $\frac{W_i}{2}$. TCP acknowledges each b arriving packets with a single ACK and whenever this ACK reaches the sender the window size is increased by $\frac{1}{W_i}$. Hence, after b windows that are fully acknowledged, the window size is increased by 1. This implies

$$W_i = \frac{W_{i-1}}{2} + \frac{X_i}{b} \ . \tag{7}$$

Similarly, the expected total number of submitted packets is

$$\begin{aligned} Y_i &= \sum_{k=0}^{X_i/b-1}\left(\frac{W_{i-1}}{2} + k\right)b + \beta_i \\ &= \frac{X_i}{2}\left(\frac{W_{i-1}}{2} + w_i - 1\right) + \beta_i \ , \end{aligned} \tag{8}$$

where β_i is the number of packets submitted in the last round before the loss indication arrived. For simplicity, it is assumed that β_i is uniformly distributed between 1 and W_{i+1}. As an approximation, it is also assumed that X_i is independent of both W_i and W_{i-1}. It then follows from (5), (7), and (8) that

$$E[W] = \sqrt{\frac{8}{3bp}} + o\left(\frac{1}{\sqrt{p}}\right) \ . \tag{9}$$

Substituting this into (1) leads to

$$B_{TCP}(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o\left(\frac{1}{\sqrt{p}}\right) \ . \tag{10}$$

For the full result and derivation of the above please refer to [19].

To analyze the behavior of FEC, we can use similar arguments, but p should be recalculated to reflect the probability that the next packet will be lost and can not be reconstructed. Denote by B the block size, $B = M + N$. Assume that k packets have already been transmitted in the current block. For the next packet to be lost without the possibility to be reconstructed, there should have been at least M loss events up to the k'th packet within the current block. Thus, the probability that the next packet is lost and is not reconstructible, given it is the $k+1$st in the block is

$$p' = p\left(1 - \sum_{i=0}^{M-1}\binom{k}{i}q^{k-i}p^i\right) \ , \tag{11}$$

where, by definition, $q = 1 - p$. For simplicity we will focus on the case $M = 1$, and will also assume that the packet's location in the block is a random variable

independent of the other variables. Therefore the average probability of a non-reconstructible lost packet is,

$$E[p'] = \frac{1}{B}\sum_{k=0}^{B-1} p(1-q^k) = p - \frac{1-q^B}{B} \ . \qquad (12)$$

Eq. (10) can now be used with $E[p']$ replacing p, but the overhead of $\frac{1}{B}$ should also be taken into account. Therefore, the bandwidth becomes,

$$B_{FEC}(p) \approx \frac{B-1}{B}\sqrt{\frac{3B}{2b(Bp-1+q^B)}} \ . \qquad (13)$$

Both Eq. (10) and Eq. (13) do not take into account the maximal window size, W_m imposed by TCP. In order to take this into account, it is noted in [19] that in the regime where $W_m \cdot p \gg 1$ the maximal window will almost never be reached and can be ignored, while when $W_m \cdot p \ll 1$, the sender will almost always work with the maximal window size. Therefore, taking into account the maximal window size,

$$B_{FEC}(p) \approx \min\left(\frac{W_m}{RTT}, \sqrt{\frac{3(B-1)^2}{2Bb(Bp-1+q^B)}}\right) \ . \qquad (14)$$

A comparison of the analytical approximation and the simulation results can be seen in Fig. 1.

If timeouts are taken into account, one can use the analysis in [19]:

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{2}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)} \ , \qquad (15)$$

where T_0 is the timeout duration (usually taken as $4RTT$). To approximate the performance of FEC we need to replace p by p'. This approximation is obtained assuming that all information in a window after a packet loss is lost. This is done only for simplifying the analysis in the timeout case, since it reduces the efficiency of FEC (since long burst errors can not be corrected).

VI. Discussion

In the previous section we analyzed the performance of FEC compared to TCP, and shown that the performance of FEC is in general substantially better. However, one may claim that the improvement in performance is due to the different policy for window management. That is, since FEC can fix up to M packet losses in a block, it does not treat these events as loss events and therefore does not decrease the window size, in case of such sporadic losses. Including a similar window management policy in an advanced version of TCP, such as TCP-SACK, will also give similar behavior of the window size, without the overhead of the extra error correction packets.

While this is true as far as the analysis above is concerned, there are other advantages to the error correction scheme over the TCP-SACK scheme (or any other TCP flavor). One advantage is the blocking of the window in TCP-SACK when a loss event occurs. If a packet is lost whose sequence number is s, during the next window transmission, no packet with sequence number higher than $s + W_m - 1$ may be transmitted. This is necessary in order to ensure that the bookkeeping is performed correctly. If the window size is currently the maximum, and the lost packet's location in the window is l, the next transmitted window cannot exceed size l, even if the window size is not decreased from hereon.

To take advantage of this property, FEC should be used in its most efficient form. The highest throughput is achieved when the transmission works at maximal window size and is kept this way throughout the process. To best utilize the error correction mechanism the block should be as large as possible. Therefore, the best performance is achieved when $N + M = W$ (W being the window size) and M is chosen such that the probability of a non-reconstructible lost packet in the window is negligible. If p is the probability of a packet loss, taking $M = 4pW$ will lead to a probability of approximately $\frac{e^{-4}}{e-1} \approx 0.01$ that a window will not be completely reconstructible. That is, 99% of the windows will reach the destination with no need for retransmission, and the overhead will amount to only $\frac{4p}{1-4p}$ of the data. The effective bandwidths then will be

$$B_{opt}(p) \approx \frac{W(1-4p)}{RTT} \ . \qquad (16)$$

For instance, if $W_m = 50$ and $p = 0.02$ (which are reasonable numbers for the Internet today) the overhead will only be $M = 4$ packets per window (approx. 8%), and the effective bandwidth will be approx. $B = \frac{46}{RTT}$, which is very close to the theoretical limit $B_{max} = \frac{49}{RTT}$ (with one lost packet event on average).

Using TCP-SACK, on the other hand, will lead to a much lower bandwidth. The first lost packet in a window is distributed according to the geometric distribution, $P(k) = (1-p)^{k-1}p$. therefore, even if we assume that the window size is never decreased, the performance will suffer from the window blocking. For the above parameters the effective bandwidth will be approximately $\frac{36}{RTT}$ (see appendix for the detailed calculation). Thus, the required transmission time will be almost 30% higher than FEC. In order to verify this, we conducted simulation of long-lived session (4000 data packets) in which the window size was never reduced. The results can be seen in Figure 6, verifying the analysis above.

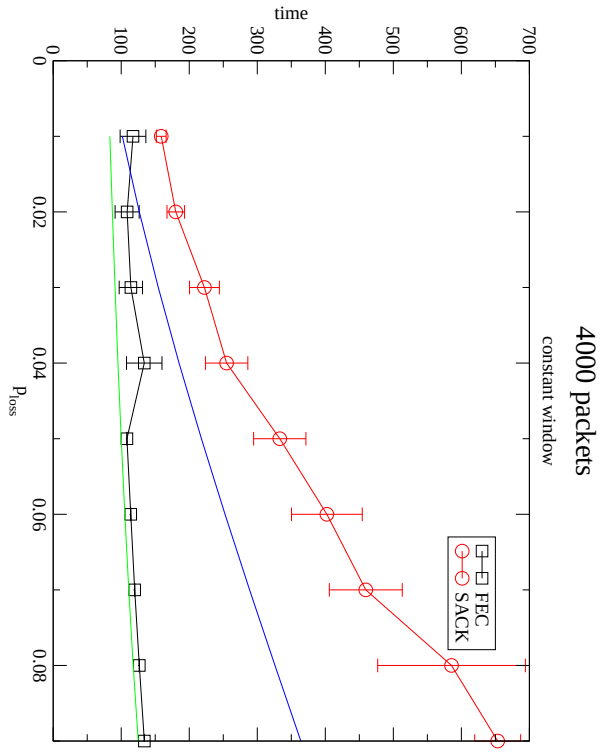

Fig. 6. Comparing Optimized Algorithms.

Another advantage of FEC over TCP is its resilience to the loss of ACK packets. There can be cases where despite the loss of ACK packets the sender can still deduce that the receiver was able to fully reconstruct the sent data. In such cases, the loss of the ACK packets will not trigger a timeout, which is a highly time consuming event (usually timeouts are only triggered after $4RTT$) and retransmission and window closing will also be avoided in such a case.

VII. Conclusions

Forward error correction is a known method for boosting network performance in the presence of packet losses. The usage of FEC within the transport layer (end-to-end manner) was already discussed and found beneficial in multicast. However, FEC for end-to-end unicast was never considered as a viable option, though its use for hop-by-hop within the link layer, is deployed, e.g., in wireless networks.

Our research results suggest that FEC can improve the overall performance and throughput for end-to-end unicast, and can be integrated within TCP. We have compared TCP integrated with FEC, denoted by TCP-FEC, with various flavors of TCP. As expected, the usage of FEC had substantially increased the performance, compared with the classical TCP (Tahoe) performance. When compared with TCP-SACK, TCP-FEC also yielded better results. Since FEC masks single lost packet (or few, depends on its parameters), we also compared it with a possible modified version of TCP-SACK which overcomes loss without closing the TCP window. This brought the performance results of both protocols to be much closer. However, the results still suggest that the packet loss under TCP-SACK prevents the window from being advanced, while the TCP-FEC totally "masks" sporadic losses and thus behaves better under such conditions.

Our studies show that the choice of right parameters affect the performance of FEC, mainly its ability to overcome sporadic packet losses obviously affects the overall throughput. An interesting research would be to try and dynamically adapt the TCP-FEC parameters as a result of changes in the network conditions. Another interesting topic is to allow the TCP implementation to decide when to "kick-in" the FEC capability in an ongoing session.

Regardless of the TCP-FEC block parameters, when the network is severely congested, or when the packet loss probability becomes high, then the TCP-FEC behaves as any other TCP flavor. It reduces the transmission rate and keeps the "friendliness" with respect to other flows in the network.

Our overall conclusion is that FEC should be considered a viable option for enhancing the performance of TCP while maintaining the important properties such as friendliness, which are crucial for the overall "health" of the Internet.

References

[1] Christian Huitema, ""the case for packet level FEC"," in *Fifth Workshop on Protocols for High-Speed Networks*, W. Dabbous and C. Diot, Eds. 1997, Chapman and Hall.

[2] R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley, MA, 1984.

[3] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, ""efficient erasure correcting codes"," *IEEE Transactions on Information Theory, Special Issue: Codes on Graphs and Iterative Algorithms*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[4] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann, "Practical loss-resilient codes," in *Proc. 29 th Symp. on Theory of Computing*, 1997, pp. 150–159.

[5] Luigi Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, Apr. 1997.

[6] G. Carle and E. Biersack, "Survey of error recovery techniques for ip-based audio-visual multicast applications," 1997.

[7] Jean-Crysostome Bolot and André Vega-Garcia, "The case for FEC-based error control for packet audio in the Internet," *to appear in ACM Multimedia Systems*, 1998.

[8] N. Shacham and P. McKenney, "Packet recovery in high-speed networks using coding and buffer management," in *IEEE INFOCOM*, 1990, pp. 124–131.
[9] Michael Luby, "LT Codes," in *Proceedings of 43rd Symposium on Foundations of Computer Science (FOCS)*, Vancouver, November 2002.
[10] Petar Maymounkov and David Mazieres, "Rateless codes and big downloads," in *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, February 2003.
[11] J. Bolot, "End-to-end packet delay and loss behavior in the internet," in *ACM SIGCOMM*, 1993, pp. 289–298.
[12] J. Postel, "Transmission Control Protocol," RFC 0793, September 1981, Internet Engineering Task Force, Network Working Group.
[13] W.R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison Wesley, Reading, MA, 1994.
[14] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated, Vol. 2: The Implementation*, Addison Wesley, Reading, MA, 1995.
[15] T. J. Socolofsky and C. J. Kale, "TCP/IP tutorial," RFC 1180, SRI Network Information Center, January 1991.
[16] E. Blanton, M. Allman, K. Fall, and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP," RFC 3517, April 2003, Internet Engineering Task Force, Network Working Group.
[17] K. Fall and S. Floyd, ""simulation-based comparisons of tahoe, reno, and sack tcp"," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.
[18] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, April 1999, Internet Engineering Task Force, Network Working Group.
[19] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose, ""modeling tcp reno performance: A simple model and its empirical validation"," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr. 2000.
[20] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast transmission," in *Proceedings of ACM SIGCOMM*, September 1997.
[21] D. Rubenstein, S. Kasera, D. Towsley, and J. Kurose, "Improving reliable multicast using active parity encoding services," in *Proc. of IEEE INFOCOM*, March 1999.
[22] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proc. of ACM SIGCOMM*, August 1998.
[23] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward Error Correction (FEC) Building Block," RFC 3452, December 2002, Internet Engineering Task Force, Network Working Group.
[24] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast," RFC 3453, December 2002, Internet Engineering Task Force, Network Working Group.
[25] "UCB/LBNL/VINT Network Simulator - ns (version 2), 1997," URL: http://www.isi.edu/nsnam/ns.

APPENDIX

In this appendix we will estimate the performance of TCP-SACK, assuming the window size is constant (i.e., the window is always kept of size W_m). If after the transmission of a window, the first n packets arrive at the destination, and the $n+1$st packet is dropped, the next window, regardless of the number of further dropped packets in the window, can not contain more than n *new* data packets (until the lost packet is recovered). Denote by $P(n)$ the (stationary) probability that a window contains n new data packets. For all $n < W_m$ $P(n)$ satisfies,

$$P(n) = \sum_{i=n+1}^{W_m} q^i p P(i) \ . \tag{17}$$

Define $F(n) = \sum_{i=n}^{W_m} P(i)$ (normalization implies $F(0) = 1$), then Eq. (17) is simplified into

$$F(n) = (1 + pq^n)F(n+1) \qquad n < W_m \ , \tag{18}$$

with solution

$$F(n) = \prod_{i=0}^{n} \frac{1}{1+pq^i} \qquad n < W_m \ . \tag{19}$$

Since $p \ll 1$ we can approximate $F(n)$ by,

$$F(n) \approx \exp\left(-p\sum_{i=0}^{n} q^i\right) = e^{q^n - 1} \qquad n < W_m \ . \tag{20}$$

For $pn < 1$ the above equation can be further approximated by $F(n) \approx e^{-np}$.

Thus, the average effective bandwidth is:

$$B(p) = \frac{1}{RTT}\sum_{n=0}^{W_m} nP(n) = \frac{1}{RTT}\sum_{n=1}^{W_m} F(n) \ . \tag{21}$$

Using the above approximations, and since $F(W_m) < 1$ (thus, its contribution is small), the effective bandwidth is approximately,

$$B(p) \approx \frac{1 - e^{-pW_m}}{p} \ . \tag{22}$$

A more accurate estimation may be obtained by numerically evaluating the sum in Eq. (21) using Eq. (19).