Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Computer Networks 46 (2004) 343–362

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Explicit transport error notification (ETEN) for error-prone wireless and satellite networks

Rajesh Krishnan [a], James P.G. Sterbenz [b,*], Wesley M. Eddy [c], Craig Partridge [a], Mark Allman [d]

[a] *BBN Technologies, Cambridge, MA, USA*
[b] *University of Massachusetts, Amherst, MA, USA*
[c] *Ohio University, USA*
[d] *International Computer Science Institute, Berkeley, CA, USA*

Available online 14 July 2004

Abstract

Wireless and satellite networks often have non-negligible packet corruption rates that can significantly degrade TCP performance. This is due to TCP's assumption that every packet loss is an indication of network congestion (causing TCP to reduce the transmission rate). This problem has received much attention in the literature. In this paper, we take a broad look at the problem of enhancing TCP performance under corruption losses, and include a discussion of the key issues. The main contributions of this paper are: (i) a confirmation of previous studies that show the reduction of TCP performance in the face of corruption loss, and in addition a plausible upper bound achievable with perfect knowledge of the cause of loss, (ii) a classification of the potential mitigation space, and (iii) the introduction of a promising new mitigation that employs rich cumulative information from intermediate nodes in a path to form a better congestion response.

We first illustrate the performance implications of corruption-based loss for a variety of networks via simulation. In addition, we show a rough upper bound on the performance gains a TCP could get if it could perfectly determine the cause of each segment loss—independent of any specific mechanism for TCP to learn the root cause of packet loss. Next, we provide a taxonomy of potential practical classes of mitigations that TCP end-points and intermediate network elements can cooperatively use to decrease the performance impact of corruption-based loss. Finally, we briefly consider a potential mitigation, called *cumulative explicit transport error notification* (CETEN), which covers a portion of the solution space previously unexplored. CETEN is shown to be a promising mitigation strategy, but a strategy with numerous formidable practical hurdles still to overcome.
© 2004 Published by Elsevier B.V.

[*] Corresponding author.
E-mail addresses: krash@bbn.com (R. Krishnan), jpgs@acm.org (J.P.G. Sterbenz), weddy@irg.cs.ohiou.edu (W.M. Eddy), craig@bbn.com (C. Partridge), mallman@icir.org (M. Allman).

1389-1286/$ - see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.comnet.2004.06.012

Keywords: Explicit transport error notification (ETEN); Explicit loss notification (ELN); Explicit congestion notification (ECN); Wireless and satellite networks; TCP performance; Congestion; Corruption; Bit errors; Channel fades

1. Introduction

The transmission control protocol (TCP) [35] is the most widely used transport protocol in the TCP/IP suite by today's common Internet users and applications. One obstacle to good performance of TCP over internetworks with wireless and satellite components is non-negligible bit-error rates (BER). TCP guarantees that corrupted data will be retransmitted by the data sender, hence providing a reliable byte-stream to applications. However, packet loss is also used by TCP to determine the level of congestion in the network [23]—as traditionally, the bulk of packet loss in networks comes from router queue overflow (i.e., congestion). Therefore, to avoid congestion collapse TCP responds to packet loss by decreasing its congestion window (*cwnd*) [4,23], and therefore, the sending rate. The reduction of the congestion window is not needed to protect network stability in the case when losses are caused by corruption and, therefore, these needless reductions in the sending rate have a negative impact on a connection's performance with little (if any) overall benefit to the network.

If a TCP sender can distinguish packets lost due to congestion from packets lost due to corruption, better performance may be achieved. The performance benefit can be realized if TCP can retransmit a packet lost due to corruption without needlessly reducing the transmission rate, while continuing to protect network stability by decreasing the sending rate when loss is caused by network congestion.

Several approaches have been proposed in the literature to distinguish congestion losses from corruption losses. For instance, methods to implicitly distinguish corruption from congestion have, thus far, not been successful [10,16]. However, performance enhancing proxies (PEPs) [12] have been shown to improve TCP performance [7], but break the end-to-end semantics of the transport layer connection. In addition, PEPs that require intrusive header inspection are not able to impact encrypted traffic (e.g., traffic utilizing IPsec [26]). Earlier work on explicit loss notification in the context of TCP over wireless and satellite links is described in [8,9,40,41]. An analysis of situations that can benefit from explicit transport error notification (ETEN) mechanisms is given in [16].

The goal and contribution of this paper is as follows. First, unlike previous work in this area, the bulk of this paper explores the problems caused by corruption-based loss and possible mitigations in a broad and generic fashion without regard to any particular mitigation mechanism. To this end, Section 2 illustrates the impact of corruption-based packet losses on standard TCP performance across a variety of network topologies and traffic patterns. Additionally, Section 2 establishes a rough upper bound on the performance a TCP can attain if the TCP can perfectly determine the cause of a dropped segment (via using an "Oracle" that knows the cause of each loss). Next, Section 3 presents a detailed taxonomy of the possible methods for mitigating the effects of corruption-based loss, including the pros and cons of various schemes. In Section 4 we depart from the broad, generic terms of the previous sections and present a preliminary examination of a novel mechanism for coping with corruption-based losses using cumulative information provided by the network. Finally, in Section 5 we conclude and summarize.

2. Can ETEN help?

In this section we present several simulations to concretely illustrate TCP's performance problems caused by corruption-based loss across a variety of network types. In addition to the impact on stock TCP, we examine a TCP variant that uses "Oracle" notifications to gain perfect knowledge about the cause of packet loss and, therefore, can mitigate the performance issues. We believe this second TCP variant, discussed in Section 2.1, is a plausible upper-bound on the performance gains a TCP could expect from a scheme to combat the issues created by corruption-based loss.

2.1. Oracle notifications

We extended the *ns-2* simulator [31] (version 2.1b9) to support our simulations. We added an "Oracle" to *ns* that sits at the end of a particular link and reports all corruption-based loss to a TCP sender. The TCP endpoint registers with the Oracle (indicating a desire to receive corruption reports) and when a corruption loss occurs the Oracle instantaneously notifies TCP of the corruption-based loss. We modified the TCP sender to record these notifications in a table for later use during loss recovery. Of course, this mechanism is not realistic, but rather the instantaneous and perfect knowledge the Oracle supplies provides an upper bound on how potential strategies to mitigate the impact of corruption-based loss *could* work.

When TCP enters its traditional loss recovery phase via fast retransmit all losses are repaired per a standard loss recovery technique (e.g., using SACK [28]). Stock TCP reduces the congestion window (*cwnd*) by half upon a fast retransmit. When using the Oracle, TCP queries the table of known corruption-based losses. If the segment being transmitted via fast retransmit was dropped due to corruption the *cwnd* is not reduced, and furthermore, a flag is set indicating the *cwnd* has not been reduced in the current window of data. If additional losses within the current window occur and are congestion-based (i.e., no Oracle notification for the loss was received) the TCP will reduce *cwnd* upon retransmission of the first congestion-based loss in the window and clear the flag that indicates a congestion response has not been invoked. This scheme is similar to using TCP SACK [11] or TCP NewReno [19] in that one *cwnd* reduction per "loss event" is taken.

In the case of loss detected via the retransmission timeout (RTO), TCP behaves the same regardless of whether Oracle notifications have arrived. In other words, Oracle notifications have no impact after an RTO. While in any given situation this is necessarily sub-optimal a clean and general approach remains illusive. Upon an RTO expiration TCP generally makes the decision that all segments sent are no longer in the network (and the SACK scoreboard is cleared). Therefore, if the sending TCP uses Oracle notifications to determine that a *cwnd* reduction is not necessary a potentially large burst of segments may be sent (bursts can *cause congestion* in some cases [22]). A second problem is that retransmission after an RTO is fairly gross with TCP often sending many more segments than necessary [3]. Therefore, in the vast majority of the cases (based on the data presented in [3]) a segment would be retransmitted for which no Oracle notification was received (and, in fact was not even lost) and, therefore, cause a *cwnd* reduction.

Finally, we note that in some cases (e.g., highly interactive traffic) the optimal response to an Oracle notification would be to retransmit the corrupted segment immediately. However, retransmission outside of a traditional TCP loss recovery period ends up having implications later in the connection due to the reordering of events. The problem stems from a retransmission being queued behind packets with higher sequence numbers. This causes the TCP receiver to transmit duplicate ACKs, which the sender, in turn, uses to detect loss. The TCP sender then needs to remember which segments have been retransmitted outside the traditional loss recovery phase and which have not. Accordingly the TCP sender must be able to determine when and if to invoke congestion control. We believe that such issues could be worked out given enough effort at redesigning TCP's traditional notions. However, in this paper we focus on bulk transfers, in which case the key objective is to keep the sending rate from being needlessly reduced. Therefore, we did not focus on optimizing when retransmits are sent with respect to the delay in getting the data to the receiver.

2.2. Single flow simulations

The first set of simulations involves a simple topology with one link between the sender and receiver. The goal of these simulations is to illustrate the impact of corruption-based loss on TCP performance, as well as to show a plausible upper-bound on the performance that could be achieved with a perfect-knowledge mitigation.

In our simulations, we use three different combinations of bandwidth and delay for the link, as follows: (i) a long–fat network (LFN) with a one-way delay of 250 ms [1] and bandwidth of 10 Mbps, (ii) a short–fat network (SFN) with a one-way delay of 25 ms and bandwidth of 10 Mbps and (iii) a long–thin network (LTN) with a one-way delay of 250 ms and bandwidth of 1.5 Mbps. All transfers are run for 30 min (ensuring that even when corruption is a very low rate event, it happens in every transfer). We applied a uniform bit-error rate (BER) of 10^{-4}–10^{-11} to the link. The highest error rate is just under 1% packet loss rate—above which TCP does not cope well. We used the *ns* standard *FullTcpSack* TCP variant. The TCP advertized window was set to 2400 segments—large enough to never be a factor in our simulations. TCP uses a segment size of 536 bytes. The capacity of the drop-tail queues applied to the link is set to the delay-bandwidth product of the network. In all the following plots the point on the far left side of the figure (at a BER of zero) is a baseline transfer with no corruption drops. In this paper we report the mean of 30 runs with each set of simulation parameters.

The simulations with BERs of 10^{-4} and 10^{-5} follow the trends shown in the following results. Furthermore, at these BERs, the difference in performance between stock SACK TCP and SACK TCP enhanced with Oracle support is nearly non-existent in all simulations presented in this section. TCP's goodput [2] at these BERs effectively makes the plots presented in this section more difficult to read by stretching the *y* axis by several orders of magnitude. Therefore, we omit these simulations from the following discussions, but summarize the simulations with the following two points: First, TCP performs quite poorly at very high BERs (often obtaining an average of less than 1 byte/s). Second, we find that the Oracle notifications do not help TCP performance in this regime due to the excessive loss and RTO behavior (including RTO backoff).

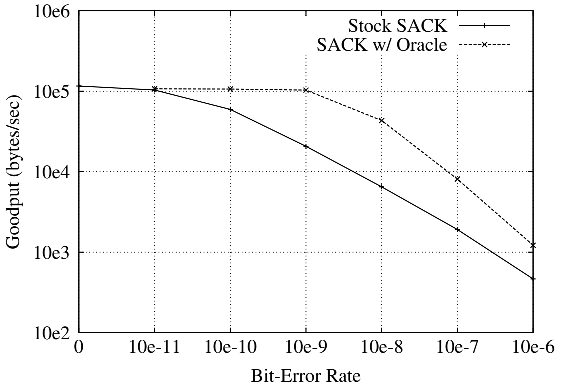


Fig. 1. LFN: Oracle vs. stock SACK TCP.

Fig. 1 shows the performance of a single TCP connection over the LFN topology as a function of the bit-error rate plotted on a log–log scale. The plot shows the general degradation of performance as the BER increases for stock TCP. The reduced performance motivates the study of mechanisms to mitigate the dramatic reduction in goodput caused by corruption-based loss. In this situation we note that even at a BER of 10^{-11} the performance of stock TCP has been reduced by roughly 10% when compared to the corruption-free case. [3]

The plot also shows that with perfect knowledge of the cause of drops TCP can improve performance dramatically. However, as the BER increases the performance suffers even with the Oracle's assistance. In this regime, the RTO plays a large part in loss recovery—which means that the perfect knowledge that has been gathered cannot be reasonably applied, as discussed in Section

[1] The propagation delay between the Earth and a geosynchronous satellite is roughly one-eighth of a second, yielding a one-way propagation delay of 250 ms and a roundtrip time of 500 ms.

[2] The *goodput* of a flow is defined as the bandwidth delivered to the receiver, excluding duplicate packets [20]. We calculate the goodput by dividing the total number of unique bytes arriving at the receiver by the duration of the TCP connection. (Note: the header bytes of these unique packets are also included.)

[3] This aspect is difficult to see in the figure due to the logarithmic scaling of the axes; we use the logarithmic scaling in order to best illustrate how the overall performance varies with BERs across several orders of magnitude.

2.1. In our LFN simulations without corruption-based loss the RTO timer never fired. On the other hand, the RTO timer expires an average of 117 times during the Oracle assisted transfers at a BER of 10^{-6} (and an average of 130 times without the Oracle).

These results suggest that mechanisms to conduct loss recovery without relying on the RTO timer when the sending rate is low would be useful. Such mechanisms would reduce the need for the gross loss recovery that the RTO timer often causes [3]. In turn, finer-grained loss recovery may help the TCP sender determine the root causes of the loss which can then aid performance. Mechanisms such as Early Retransmit [2] and Smart Framing [30] may be useful in this space and warrant further study.

Fig. 2 shows the performance of a single TCP connection over the SFN topology as a function of the BER on a log–log plot. When compared to the LFN simulations presented above, the SFN plot shows that the shorter RTT of the network aids TCP performance by tightening the congestion control loop. Stock SACK's performance first drops below full utilization (by roughly 85%) at a BER of 10^{-8} in this set of simulations—much later than the BER of 10^{-11} where the drop-off first occurs in the LFN case presented above. Additionally, we see the performance at the worst BER is an order of magnitude better than the same point in the LFN simulations. While the shorter feedback loop aids TCP performance, the impact of corruption-based loss is still significant (over an order of magnitude difference at high error rates). Finally, in these experiments we again observe the power in being able to determine the cause of each packet loss and how that power is diminished as the connection starts to rely on the RTO for loss recovery.

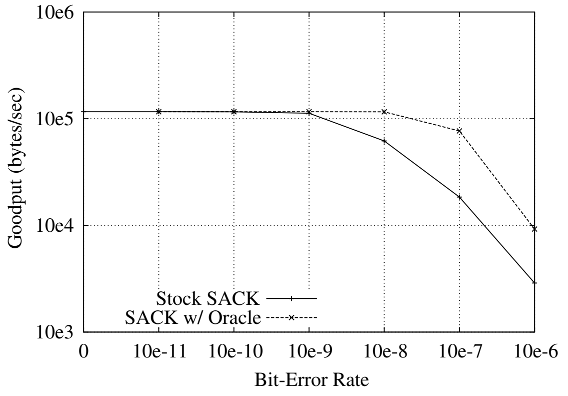


Fig. 2. SFN: Oracle vs. stock SACK TCP.

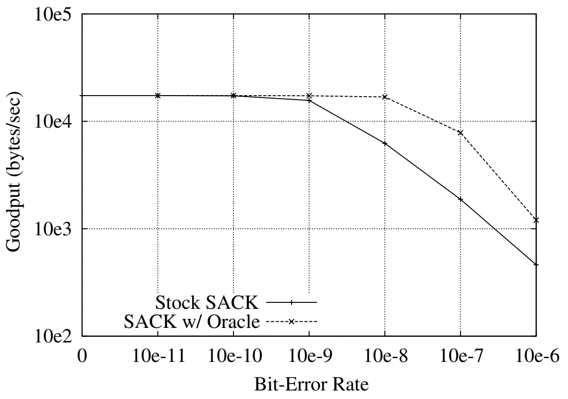


Fig. 3. LTN: Oracle vs. stock SACK TCP.

Finally, Fig. 3 shows the performance of a single TCP connection over the LTN topology as a function of the BER plotted on a log–log scale. In this plot we see that TCP has lower goodput due to the smaller amount of capacity on the bottleneck link than used in the LFN set of simulations. However, we also note a similar decline in performance as the BER increases, as we have illustrated previously. Further, with the Oracle's help the performance is significantly improved over stock TCP—again suggesting that mechanisms that offer TCP more information about the cause of losses would be worthwhile to bulk data transfer applications.

2.3. Competing traffic

To verify that the above results hold in a slightly more practical environment our next set of simulations involves competing traffic. While this simulation is still not a realistic Internet setting, it gives a glimpse of how TCP copes with corruption-based loss when there is also contention for bottleneck resources between various traffic flows. The simulations presented in this section involve a four node topology with a TCP source and TCP destination separated by two routers. The link between the end nodes and the routers has a capacity of 10 Mbps and a one-way delay of 1 ms. The link between the routers has a capacity of 1.5 Mbps, a one-way delay of 250 ms and router queue sizes set based on the delay-bandwidth product of the path (these are the same settings used for the LTN experiments outlined above and shown in Fig. 3). The competing traffic consists of four constant-bit rate on/off UDP flows in each direction over the bottleneck link (between the routers). The on and off times of the flows are dictated by an exponential random process with mean on and off times of 0.5 s. When on, each flow sends at 0.25 Mbps. When all competing flows are active they consume two-thirds of the bottleneck capacity. The first UDP flow in each direction is started 60 ms into the simulation, with an additional UDP flow starting in each direction every 50 ms (until four on/off flows are active in each direction).

Fig. 4 shows the average goodput of the end-to-end TCP connection over 30 simulation runs as a function of the BER on a log-log plot. The figure shows the same general trends illustrated in the single connection LTN case. The impact of the bursty on/off traffic is to reduce the available bottleneck capacity by roughly one-third. [4] The figure shows that corruption-based loss negatively impacts stock TCP performance in a scenario with competing traffic. Further, the figure shows that with perfect knowledge a TCP sender can enjoy performance benefits across a range of BERs, but the benefits diminish as the BER increases and TCP relies more heavily on the RTO for loss recovery.

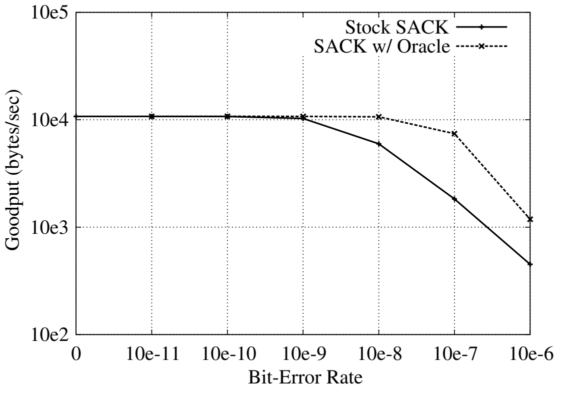


Fig. 4. LTN with competing traffic.

2.4. Discussion

The results in this section confirm previous work (e.g., [8,9,40,41]) in showing that schemes that allow a sending TCP to determine the cause of a segment loss would be useful to bulk transfer applications, especially in networks with non-negligible packet corruption rates. This conclusion holds across a number of different network types and a range of BERs. We classify the solution space for mitigations in the next section.

3. A taxonomy of corruption notification and response mechanisms

In this section we present a taxonomy describing the range of mechanisms that can be used for loss discrimination, explicit transport notification, and mitigation. First, we offer the following definitions to clearly distinguish different transport protocol mechanisms, as illustrated in Fig. 5:

- Flow control is exerted by the *receiver* to prevent the sender from transmitting data at a rate that exceeds the capacity of the receiver.

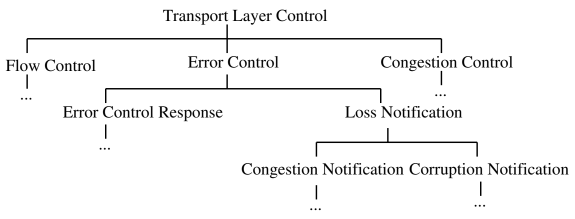


Fig. 5. Transport layer control.

[4] The UDP flows are expected to consume one-third of the capacity since the flows are set up to consume two-thirds of the bottleneck capacity when all flows are sending at the same time and the flows are configured to send roughly half the time.

- Congestion control [15] and avoidance is used to prevent the sender from transmitting data too quickly for the *network* to handle.
- Error control is a function needed for the reliable delivery of data; this function is responsible for retransmitting information that is lost (due to either corruption or congestion) between the sender and receiver.

In this paper we are concerned with error control, in particular discriminating loss that is caused by corruption from loss caused by congestion. Congestion-based losses are caused by resource contention or control in networks. For instance, packets arriving at a router that has exhausted its buffer memory may be dropped—indicating contention caused by a mismatch in the packet arrival and packet departure rates at the router. In this paper we will use the term *congestion loss* to refer to packets not arriving at their destination due to resource contention somewhere along the path.

Corruption is generally caused either by channel errors (such as background noise or interference) or by hardware errors in network components [44]. Corruption can consist of bit errors, packet loss, or burst errors, depending on the duration of a particular error event. We will use the term *corruption loss* to refer to packets that do not arrive intact at their destination due to the information contained in the packet (either header or payload) being unexpectedly changed [5] during transit.

3.1. Loss discrimination

Loss discrimination refers to determining whether a packet loss event was due to corruption or congestion. We define two major classes of loss discrimination: implicit and explicit.

3.1.1. Implicit loss discrimination

Implicit loss discrimination does not rely on mechanisms that definitively identify the causes of packet losses. Rather, implicit discrimination mechanisms make assumptions on the cause of loss to determine the appropriate error, flow, or congestion control response. This inference can span the range:

- All losses are due to congestion; this assumption is valid in networks that are engineered to have highly reliable links, and is generally valid for wired networks. This is the assumption that TCP makes and has prevented congestion collapse in the traditional wired Internet. This assumption is conservative in that it errs on the side of protecting the network at the expense of performance when loss is not caused by congestion.
- Losses may either be due to corruption or due to congestion, or both. It may be possible to use additional information (e.g., grouping of packet losses, and delay variations) to better infer the cause of loss. For example, networks that use a different form of congestion control than TCP's loss-based scheme (e.g., delay-based congestion control [13] or congestion control that relies on explicit information from the network [25]) could enable such inference.
- All losses are due to corruption; this assumption is valid in lossy networks where there is no chance of congestion, either due to overprovisioning or guaranteed resource reservation.

Previous work (e.g., [10]) concluded that implicit loss discrimination is not an effective strategy. However, congestion avoidance behaviors based on accurate estimation of the end-to-end path capacity can enhance TCP performance in certain environments in which losses can occur both due to congestion and corruption. Examples of congestion avoidance behaviors that implicitly account for corruption losses based on path capacity estimation include TCP Westwood [14] and TCP Peach [1].

TCP Westwood is a sender-side modification to TCP Reno that continuously estimates the bottleneck capacity for the end-to-end path (based on the times when acknowledgments are received), and adjusts the congestion window based on the estimated capacity [14]. Since packets dropped

[5] Some packet transformations, such as TTL reduction, are expected and are not considered to be packet corruption.

due to corruption should not reduce the estimated capacity (assuming accurate measurements and estimation), the loss discrimination is, therefore, *implicitly* included in the congestion response.

TCP Peach, a congestion control scheme proposed for satellite networks, uses dummy segments (that must be treated as low-priority segments by all intermediate nodes) to probe the availability of network resources [1]. If all the dummy segments are acknowledged, then the sender interprets this as evidence that there are unused resources in the network and accordingly can increase its transmission rate. In TCP-Peach, corruption errors are not explicitly notified, but instead *implicitly* accounted for by the capacity estimation strategy.

3.1.2. Explicit loss discrimination

Explicit loss discrimination is based on mechanisms that explicitly signal loss due to corruption, congestion, or both.

It is important to note that corruption cannot be directly inferred from explicit congestion notification (e.g., ECN [21]), and vice versa. This is due to the fact that a given packet may experience *both* congestion as well as be dropped due to corruption. Furthermore, in cases where these mechanisms are cumulative or statistical in nature, it becomes more difficult to infer one from the other.

In this paper we focus primarily on explicit loss discrimination. We present a taxonomy for *explicit transport error notification* (ETEN) mechanisms next. We examine ETEN mechanisms along two orthogonal axes, namely, node behavior and control loop issues.

3.2. ETEN node behavior

There are two classes of behavior of concern to ETEN: notification and response. This is reflected in the behavior of two types of nodes:

1. The *sender* is the transport endpoint that transmits data, and is typically responsible for response behavior. In the case of reliable end-to-end communication, this is the node that will be required to retransmit data that has not successfully reached the receiver. In the case of TCP, the sender is also responsible for congestion control decisions. [6]
2. The *notifier* is a node that detects a corruption event and initiates a notification that will ultimately reach the sender. The notifier may involve the receiving node, or the intermediate nodes along the communication path.

Note that in this work we are concerned with only corruption losses that are end-to-end in scope. Generally speaking, mechanisms that attempt local recovery of lost packets and try to hide those losses from the sender are out of scope for this paper. In particular, link-layer retransmissions, link-layer forward error correction (FEC) and performance-enhancing proxies [12] (e.g., *snoop* [9]) may be used in conjunction with the mechanisms involving the end-hosts discussed in this paper, but are specifically out of scope for our discussions.

The sender and notifier nodes each exhibit *observation*, *decision*, and *action* behaviors, discussed briefly in the following subsections.

3.2.1. Notifier behavior

The notifier, as defined earlier, is either an intermediate or receiving node that detects corruption and is responsible for acting in a manner that will ultimately notify the sender.

Notifier observations consist of detecting corruption events, for example due to a checksum calculation or feedback from the link layer.

Notifier decisions determine when and how to make corruption notifications. For example, in the case of cumulative ETEN the notifier will have to determine the time interval over which to compute corruption statistics and the times at which the notifications should occur. If multiple mechanisms are in effect, the notifier must decide which is the appropriate one to use.

Notifier actions are the signaling mechanisms used to report corruption-based loss. This may range from sending an explicit ETEN signaling

[6] The sender ultimately controls the data transmission rate and so is always at least a *component* of congestion control.

message directly back to the sender on the detection of a corrupted packet (out-of-band backward packet-granularity ETEN) to modifying a header field that is accumulating path corruption statistics (in-band forward cumulative ETEN). Notifier action might also consist of dropping a corrupted packet or merely marking it as corrupt as it is forwarded. The range of actions is discussed further in Section 3.3.

3.2.2. Sender behavior

The sender is the node that will have to take actions to retransmit data once it has been notified.

Sender observations consist of understanding corruption signaling from the notifier (whether as explicit ETEN signaling messages or embedded in returning acknowledgments), congestion information (whether explicitly signaled as in ECN or inferred as in the lack of an acknowledgment), as well as local observations on its own environment, such as offered load.

Sender decisions determine what action should take place based on notification and other observations, for example the time and granularity of retransmissions. A key additional decision is the determination of the likelihood that a given loss event is due to congestion, particularly in the absence of explicit congestion notification. As mentioned earlier, this cannot be correctly inferred in the absence of an ETEN notification, since a given loss event may be due to both corruption and congestion.

Sender actions are simply the actions taken in response to corruption, including packet retransmission and dynamic FEC strength adjustment. Additionally, sender actions include the appropriate congestion control action, such as throttling the sender's transmission rate.

The next section describes various control mechanisms that can be applied to the notifier–sender control loop. In some cases sender and notifier behavior are highly dependent on one another. For example, if the notifier uses out-of-band backward ETEN signaling messages to indicate corruption, the sender must be capable of receiving and parsing the messages. In other cases, the notifier and sender may operate independently. For example, the granularity of corruption notification may be smaller than, equal to, or larger than the granularity of sender retransmission.

3.3. Control loop

Corruption notification and response involves a control loop between the *notifier* nodes that are involved in the detection and notification of corruption and the *sender* of information that must respond in order to enable recovery from the corruption losses. The notifiers may be intermediate network nodes, the receiver, or both.

In the following subsections, we describe in detail the various aspects of this control loop, namely: (i) feedback, (ii) locus, (iii) granularity, (iv) in- vs. out-of-band signaling, (v) direction of control information flow, and (vi) determinism.

We illustrate the taxonomy from the perspective of the response in Fig. 6, and provide a notification-centric perspective in Fig. 7.

3.3.1. Feedback

The ETEN *feedback* loop can be open, closed, or a hybrid.

Closed-loop feedback requires that acknowledgments (positive or negative) are returned to the sender to indicate which packets have been received intact and which have been corrupted. [7] This is typically an ARQ mechanism with a number of possible variants such as go-back-*n* and selective repeat.

Open-loop feedback uses forward error correction (FEC) to provide statistical guarantees on a packet's successful transmission. Often FEC schemes are tightly coupled with a particular channel corruption model.

Hybrid open/closed-loop feedback combines both mechanisms: open-loop FEC to reduce the need for acknowledgment-based retransmissions, with acknowledgments as necessary to trigger retransmits and *guarantee* the delivery of data (or, at least an understanding by the sender that the data were not successfully delivered).

[7] The sender must have some default behavior to avoid becoming deadlocked if an acknowledgment does not arrive (e.g., a timeout with a default assumption about the cause of loss).

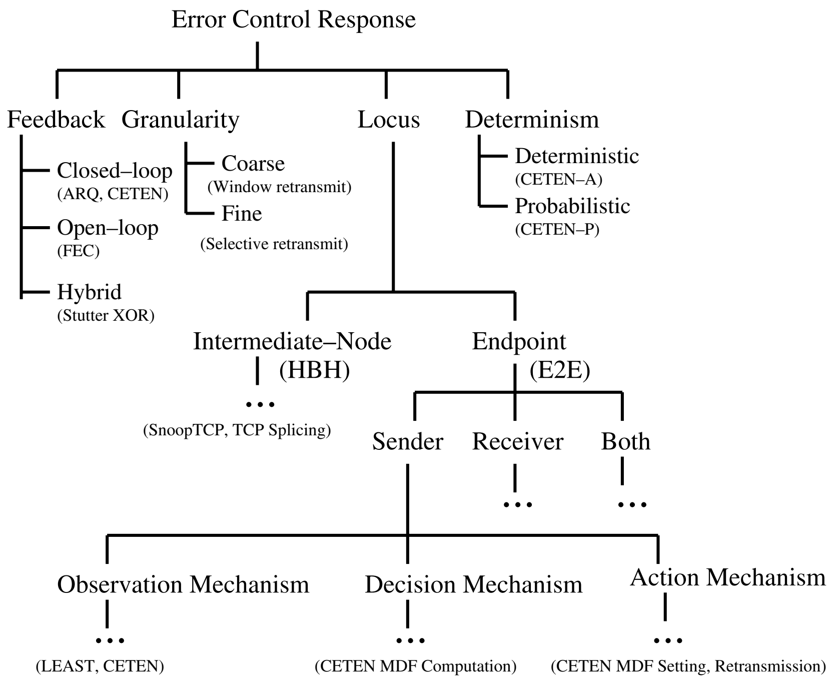


Fig. 6. Error correction response.

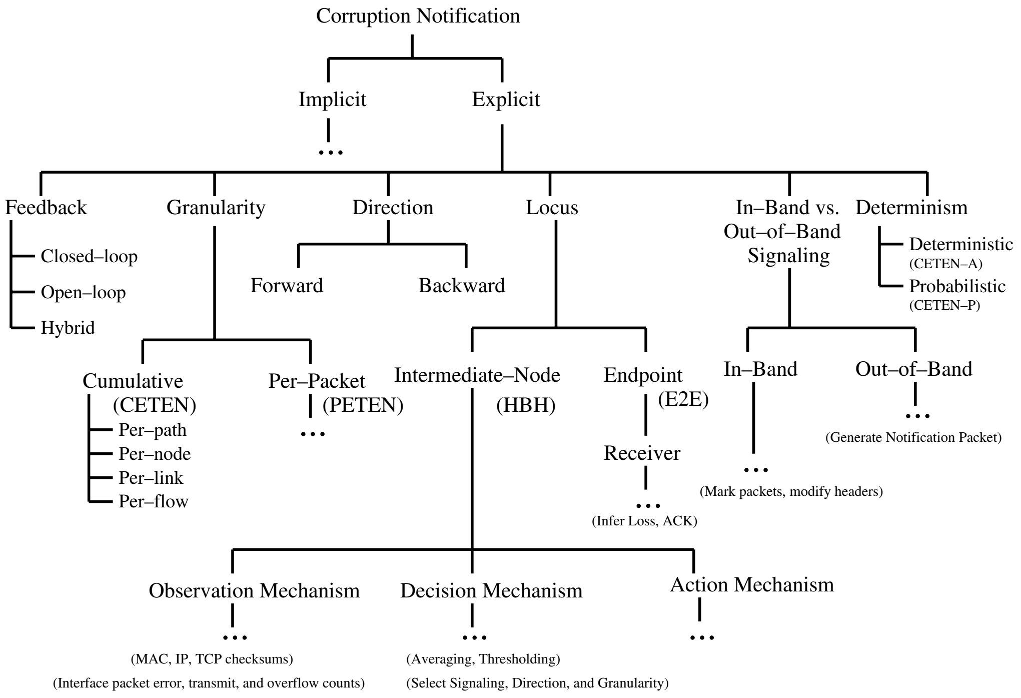


Fig. 7. Corruption loss notification.

There are fundamentally two ways in which FEC strategies can be used for ETEN: either the error correction code can be contained entirely within each packet or it can be distributed across multiple packets. In the first case, each packet can include additional bits of error correcting information; intermediate nodes can detect and if possible correct corruption before forwarding the packet. A large number of error correcting codes that are effective under different error models are available.

In the second case, erasure codes can be used that allow corrupted packets to be dropped while allowing the end points to recover the information from additional redundant packets. The Stutter XOR scheme [24] is an example of a simple erasure code. More sophisticated codes have been applied to packet-switched networks [29,39,43].

3.3.1.1. Deployment challenges for FEC schemes with TCP/IP. There are significant challenges to combining FEC with some form of ETEN for TCP/IP. Any *reliable* transport protocol must still provide end-to-end ARQ to guarantee packet delivery. TCP, in particular, uses ARQ in its combined error, flow, and congestion control algorithms; the addition of, and interaction with, FEC may add significant protocol complexity.

In the case of satellite or wireless links, per-packet FEC cannot protect against all non-congestion packet losses, for example, channel fades. Furthermore, IP routers simply drop erroneous packets to prevent mis-forwarding [6]. With per-packet FEC, intermediate IP routers would be required to correct packet headers (provided there is no IP–IP encapsulation, else the payload may also have to be corrected at intermediate routers) to ensure misforwarding. Furthermore, for any given path MTU, the use of variable strength FEC means that the MSS seen by TCP will fluctuate with the corruption rate.

The interactions of end-to-end TCP mechanisms for flow control, loss recovery, and congestion avoidance with erasure codes is much more subtle. There is tension between erasure codes, on the one hand, trying to mask all packet losses (whether due to congestion or corruption including fades) and prevent retransmissions, and TCP on the other hand, relying on the congestion losses to provide feedback to congestion avoidance mechanisms. This masking of losses challenges the fundamental ETEN goal of being able to discriminate between corruption and congestion packet losses.

TCP ACKs carry the sequence number of the next byte of data the receiver expects to arrive. This allows the sender to determine packet losses and adjust the congestion window. When erasure codes are used, this feedback is insufficient since the last segment being accounted for (as received) may belie the fact that some packets could have been lost due to congestion (but were reconstructed at the receiver). Enough packets must be dropped so as to exceed the capability of the code before the TCP sender is actually notified of congestion. This added delay might make the congestion avoidance loop unstable.

Solving this problem requires that we keep track of not only the sequence number of the payload data but also the sequence number of the encoded packets. In this case, TCP congestion avoidance could use this latter sequence number. This will require the addition of this information to the IP or TCP packet headers (perhaps in the form of an option).

Furthermore, with erasure codes, the receiving TCP has to wait for the possibility of subsequent packets correcting a loss. This can conflict with the settings of the retransmit timer and the delayed acknowledgment timer.

3.3.2. Locus of ETEN

We use *locus* of control to describe the span of the ETEN control loop, in particular to define the *notifier* node or nodes that are responsible for corruption detection and reporting back to the *sender*.

End-to-End (E2E) ETEN relies only on the receiver to serve as the notifier that detects corruption and informs the sender.

Hop-by-Hop (HBH) ETEN relies on nodes along the path to serve as notifiers to detect and report corruption. HBH schemes involve the intermediate network nodes (switches or routers) as well as the receiver (for the last hop). Additionally, the receiver will be involved in any necessary end-to-end recovery notification, including relaying forward ETEN messages to the sender (as discussed in Section 3.3.5). Note that even though we generally think of TCP/IP as having only end-to-end loss recovery, the IP checksum and IP router semantics that require the dropping of corrupted packets [6] is a HBH component of the TCP/IP loss recovery process.

From a deployment perspective, ETEN mechanisms that allow *selected* intermediate nodes in a path to participate in the corruption detection and notification scheme are more desirable than those ETEN mechanisms which require *all* intermediate nodes in the path to participate. The former has the significant practical advantage of allowing selective deployment of nodes that need corruption notification mechanisms rather than requiring massive replacement of network infrastructure. For example, candidates for the deployment of ETEN notifier nodes are wireless access points and gateways, and switches that terminate long-haul wireless and satellite links.

3.3.3. Granularity

The *granularity* of ETEN corruption feedback refers to the scope over which corruption detection, notification, and response actions are taken. At the highest level, we refer to the granularity as either per-*packet* (PETEN) or *cumulative* (CETEN).

Packet-based (PETEN) mechanisms are able to detect, report, and respond to individual packet corruption events. Per-packet notifiers are able to properly convey the fact that *individual* packets have been corrupted; per-packet senders are able to retransmit those (and only those) packets that require retransmission. The Oracle ETEN described in Section 2.1 is a PETEN with the ability to perfectly determine addressing and sequence numbers for each packet.

PETEN requires not only that the sender and notifier perform corruption detection and notification on a per-packet granularity, but that the notifiers that detect corruption are able to properly identify corrupted or obliterated packets. Thus, the source and destination address as well as the sequence number must be available or reconstructed. In the case of TCP, this consists of the source and destination IP addresses, the source and destination TCP ports, and the TCP sequence number. In addition, the packet in question must be part of the sender's current window; otherwise, the opportunity to mitigate the performance problems caused by the corrupted packet is lost.

In practice PETEN may be challenging since it requires that the notifier have a reliable mechanism with which it can determine the transport endpoints. One solution to consider is to separately protect the header by a strong FEC check. Another is to obtain this information from link layer recovery mechanisms (e.g., the upstream neighbor that had to retransmit a packet can generate such notifications). In the absence of such mechanisms, observations and notifications of corruption loss have to be at a coarser granularity, described next.

Cumulative ETEN (CETEN) mechanisms are needed when the notifier nodes can only calculate cumulative corruption rates for each link. In other words, the information in the header of a corrupted packet is considered inaccurate and cannot be constructed with enough confidence to allow PETEN mechanisms to perform well.

The cumulative CETEN information conveyed to the end-hosts can be in one of several different forms:

- An *absolute* corruption rate (bit-based, byte-based or packet-based) observed within a moving window in time. The corruption rate may be quantized into a small number of steps (for example, *high*, *medium*, and *low*). A binary feedback scheme [38] (see also [36,37]) is a special case that provides indication that the bit/byte/packet corruption rate exceeds some threshold.
- A *relative* corruption rate that simply indicates that the quantized corruption rate has increased or decreased from the previous value.
- An estimate of the probability that a packet survives corruption.

There are various possibilities for the aggregation of the cumulative corruption statistics from each notifier (e.g., per-flow, per-path, per-link, or per-node). Furthermore, CETEN information can be collected on a per-hop basis or aggregated over the end-to-end path. Due to the difficulty in correctly assigning corrupted packets to their corresponding flows, any per-flow CETEN information has to be estimated, for example from what is observed across all flows using a given link.

Estimating and correctly attributing the fraction of the observed aggregate corruption loss rate on a per-flow basis can add significant complexity to the node (except perhaps at the receiver). Determining whether this can be done reliably (and if so, how) requires further study. We investigate CETEN further in Section 4.

The applicability of PETEN and CETEN mechanisms to various application and network scenarios under various error models also requires further study.

3.3.4. In-band vs. out-of-band signaling

ETEN signaling can either be *out-of-band* or *in-band.*

Out-of-band (OB) signaling uses distinct ETEN signaling messages (e.g., using ICMP) that are propagated from the notifier node to the sender (either backward or forward, as described in the following subsection).

In-band (IB) signaling modifies or piggybacks on the headers of data packets and acknowledgments. In-band signaling is particularly attractive for CETEN schemes that propagate corruption statistics in the packet header. In this case, each CETEN-capable intermediate notifier node modifies the corruption rate carried in the packet header, so that when the packet reaches its destination the receiver knows the path corruption rate.

Note that for packet-based PETEN, if corrupted packets are dropped (as in IP [6]), the ETEN indication must be contained in other packets belonging to the same flow. Alternatively, if the packet header is separately protected by an error check and only the payload is corrupted, the packet could be marked as corrupt and forwarded towards the destination.

3.3.5. Direction of notification

Notifications can either be sent directly back to the sender, or proceed to the destination to be returned to the sender.

Backward ETEN propagates notifications backward, analogous to backward explicit congestion notification schemes (e.g., source-quench [34] and ATM BECN [5]). In these cases notifiers use out-of-band signaling messages destined to the sender.

It is also conceivable to piggyback backward ETEN information in returning acknowledgments to the sender (i.e., in-band backward ETEN), but this adds significant complexity to the notifier.

Forward ETEN propagates notifications forward to the destination, analogous to forward explicit congestion notification schemes (e.g., ATM FECN [5] and IP-based ECN [21,36,37]).

If separate messages are generated per packet corruption loss, it is easy to see that backward PETEN could lead to faster loss repair than forward PETEN. The potential performance benefit of using backward ETEN is higher if the corruption occurs closer to the sender and increases with the round-trip delay of the path.

Two in-band signaling alternatives that do not require generation of new packets for forward ETEN exist. With the first alternative, the intermediate notifier node that detects a corrupted packet can convey this information by marking or modifying headers of subsequent packets. If reliable per-flow assignment of the corruption is possible, then this operation can be restricted to subsequent packets belonging to the same flow. This requires the maintenance of sufficient per-flow state to find a subsequent packet on the same flow. The other approach is to forward the corrupted packet (suitably marked or encapsulated) and pass it along to the destination (subsequent nodes must also forward this packet), rather than dropping it (as currently required by IP router semantics [6]). The destination in turn can notify the sender of the packet lost due to corruption.

3.3.6. Determinism

The last aspect of the ETEN control mechanism to consider is how deterministic actions are.

Deterministic actions are used when a particular response is needed and sufficient knowledge is available. An example of deterministic action by the notifier is the transmission of a backward PETEN message for a corrupted packet from which the header could be correctly decoded. A deterministic sender response would be to retransmit this packet.

Probabilistic actions are taken based on information that is statistical or inferred without certainty. An example of probabilistic notifier behavior is transmission of a backward PETEN message when the header cannot be fully reconstructed (but perhaps inferred with reasonable confidence based on comparing the corrupted packet's header with collected per-flow state). An example of sender probabilistic behavior is adjusting the congestion window a fraction of the time based on an estimate of the fraction of losses due to congestion (as will be described in Section 4).

In this section, we provided a taxonomy of the ETEN solution space. The key issues are:

- where, how, and what information about corruption is observed and tracked by the notifier;
- how does the notifier decide on when and by what means to convey the information to the sender;
- what information related to loss recovery does the sender track and how;
- how does the sender decide how to discriminate among losses, and by what means to recover from losses;
- design of mechanisms for detection, notification, and response to corruption losses.

We discussed that various alternatives exist for each one of these issues. The potential gains in Section 2 motivate further exploration and evaluation of the alternatives, in terms of how well they perform and how best to combine them into an end-to-end solution. In the next section, we present a promising new CETEN approach that combines particular approaches within this space.

4. Cumulative ETEN

The last two sections of this paper have broadly and generally discussed the implications of corruption-based loss on TCP performance and what mechanisms could be used to counteract the impact of corruption-based loss. In this section we narrow our focus to a novel class of mitigation for combating the impact of corruption-based loss. In this section we explore cumulative explicit transport error notification (CETEN) techniques that are applicable when sufficient information about the cause of specific packet drops is not available to the transport layer endpoints. Rather, using CETEN the TCP sender relies on corruption rate statistics provided by the network to drive the behavior of the congestion control algorithms. In this section, we describe two CETEN strategies and present a brief set of simulations that show their promise. The CETEN presentation in this paper is preliminary and meant to suggest a new mechanism that attempts to achieve the ideals presented in Section 2. More in-depth treatments of CETEN issues are provided in [17,18].

4.1. Determining the packet corruption rate

The first problem we tackle is that of transmitting rich information about the corruption rate detected within the network to the transport endpoints. The mechanism we employ in our study adds a *corruption survival-probability* field to each packet. This value represents the probability that a packet avoids corruption as it traverses the network path. The survival probability field is initialized to 1.0 by the source of the packet and is updated by intermediate nodes along the path (as described in more detail below). When a packet arrives at the receiver the survival probability contained in the packet is the survival probability of the entire path. The transport endpoint at the destination keeps a record of the survival probability of the forward path and echoes the probability back to the sender in the next ACK packet transmitted. As discussed in Section 3 there are alternative methods for gathering the information. Experimenting with those methods is left as future work.

Each intermediate node in the path is responsible for tracking the corruption rate, r, on their incoming links. [8] Each intermediate node then multiplies the path corruption survival probability field from each packet header by the node's own estimate of the link corruption survival probability, $(1-r)$, for the link on which the packet arrived.

[8] In practice, we only expect intermediate nodes connected to links experiencing non-negligible amounts of corruption to implement CETEN. An intermediate node that does not experience corruption loss will essentially not change the path state and, therefore, the work involved would be wasted effort.

The exact method for arriving at the link error rate is a subject for future work. In the experiments presented in this paper r represents the configured link corruption rate rather than a corruption rate that is tracked over time. Using the configured corruption rate as r allows us to assess the upper bound on the performance improvements that are possible without any estimation error. Designing methods to track the corruption rate is clearly a rich area of future work. Possible schemes for arriving at error rates (and smoothing/averaging them over time) are limitless. A possible approach is given in [27]. Finally, we verified the observed corruption rate to be within 10% of the configured corruption rate in our simulations.

We note that there is a delay between an intermediate host noting its corruption rate and the sender ultimately receiving that information. The delay is less than the RTT of the network path. We believe this delay is tolerable given that we envision the intermediate node reporting corruption rates somehow averaged over a number of RTTs. However, if corruption rates are to be reported for shorter time intervals then the delay in getting the information to the TCP sender may play a part in the overall effectiveness of CETEN. Such a scenario is not explored in this paper and is left as future work.

4.2. Computing the total loss rate

Loss can be either due to congestion or corruption. [9] In theory, if a TCP knew how to ascertain the fraction of losses due to one cause (say, losses due to corruption, as outlined above) and if the TCP can determine the total loss rate, then the TCP can determine the losses due to the other cause. A natural method for ascertaining the total loss rate is for the TCP sender to count the number of retransmissions. However, as shown in [3] this method ends up significantly overestimating the total loss rate due to TCP's sometimes gross retransmission strategies. A family of algorithms (called *LEAST*) is presented in [3] that TCP senders can use to estimate the total loss rate to within 10% of the actual loss rate in over 90% of the TCP connections studied (using the NIMI mesh of Internet measurement points [32]).

An alternative approach to estimating the total loss rate is to have the network inform the TCP endpoint about the current congestion-survival probability, much like the scheme outlined above for corruption information; [27] outlines such a scheme. In addition, the XCP congestion control technique [25] could also be leveraged to help disambiguate the cause of losses. The biggest weakness of such an "in-the-network" scheme is that if some congested routers do not participate they cause the sender to overestimate the fraction of losses attributed to corruption (by underestimating the congestion rate) and, therefore, inject more traffic into the network than appropriate. In-the-network strategies require less accounting on the part of the TCP sender/receiver; however, there also could be issues relating to the soundness of the estimates of corruption and congestion in the network.

For the work presented in this paper we use the *LEAST* loss estimation technique in the TCP sender to estimate the total loss rate when needed.

4.3. Alternate congestion responses

In this section, we address the question of what the sender could do with the corruption probability estimates and how TCP's congestion response may be changed to incorporate this new information. We specify two different schemes that could be used by a TCP sender to mitigate the performance impact of corruption. These are far from the only two schemes that could be used. However, determining the best variant for general use is beyond the scope of this paper.

4.3.1. Probabilistic: CETEN$_P$

Given that TCP has inferred loss(es) from duplicate acknowledgments [4], selective acknowledgments (SACKs) [28] and/or retransmission timeouts [33] TCP needs a way to decide on a

[9] Exactly how to handle the case described in the last section when a packet experiences both congestion and corruption is outside the scope of this paper. Also, in our simulations loss *is* either caused by congestion or corruption and never crosses into this gray area.

congestion control response. For the CETEN$_P$ variant we use a weighted coin flip based on the estimated fraction of the losses due to corruption, $\frac{e}{p}$, where e is the fraction of packets dropped due to corruption and p is the fraction of packets dropped for any reason. If, probabilistically, a particular loss is attributed to packet corruption the lost segment can be retransmitted without modifying the congestion control state. Otherwise, the TCP retransmits the lost segment and invokes standard congestion control procedures (i.e., reducing the congestion window by half). While CETEN$_P$ may not correctly choose whether to change TCP's congestion control state on any particular loss, the goal is to provide the appropriate *average, long-term* congestion response without incurring the traditional susceptibility to losses caused by corruption.

4.3.2. Adaptive adjustment: CETEN$_A$

An alternative to the binary decision with regards to invoking congestion control offered by CETEN$_P$, CETEN$_A$ provides an adaptive scheme that reacts to each loss, but not by using the traditional multiplicative decrease factor (MDF) that stock TCP uses (one-half). Rather, CETEN$_A$'s MDF is defined as:

$$\mathrm{MDF} = \frac{1 + \left(\frac{e}{np}\right)^k}{2}, \quad n \geqslant 1,\ k > 0,\ p > 0, \tag{1}$$

where p is the total packet loss rate, e is the corruption loss rate and n and k are parameters that allow for the shaping and bounding of the MDF. In the experiments presented in this paper we use $n=k=1$ which provides a congestion response as if the only losses were those caused by congestion. When $n=k=1$ and all loss is caused by congestion the standard MDF of one-half is used. However, if all loss is due to packet corruption an MDF of 1 is used (i.e., no *cwnd* reduction). Varying n and k can make the response more conservative (or more aggressive) and likely has implications on fairness. Future work should include experimenting with these shaping parameters, but such work is beyond the scope of the initial evaluation presented in this paper. Finally, note that any continuous monotonically increasing function based on $\frac{e}{p}$ that is no more aggressive than Eq. (1) with $n=k=1$ can be used to determine the MDF.

4.4. CETEN simulations

To investigate CETEN we implemented both CETEN$_A$ and CETEN$_P$ in *ns-2*. CETEN is implemented in the *ns sack1* TCP variant rather than the *FullTcpSack* variant used in Section 2 because *sack1* supports DSACK (which is needed for the estimate of p). The simulations consist of a four-node network with TCP end points separated by two routers. The routers are connected to each other with a 5 Mbps link with a 40 ms one-way propagation delay. The routers use drop-tail queues with 150 packet buffer sizes. A uniform random process is used to insert corruption-based drops on the link between the routers. The corruption-rate is varied (as shown in our results). Each host is connected to a router via a 10 Mbps link with a one-way propagation delay of 3 ms. The TCP endpoint uses an advertized window of 500 segments—enough to never be a performance issue in our simulations. The hosts use an MSS of 1460 bytes and delayed ACKs. This scenario is different from the scenarios used in Section 2. The TCP sender estimates the total loss rate using the DSACK version of the *LEAST* algorithm [3]. This simulation setup allows for the TCP to self-congest the network (i.e., a single TCP connection can consume the network capacity and the entire router queue causing congestion-based losses to occur). All simulations are run for 1 h to assess the long-term average sending rate. The following results represent the average of 30 random simulations.

The situation presented in this section is more akin to a terrestrial wireless network than those previously explored in Section 2. Since the TCP model is generally discussed in terms of packet loss rates rather than bit-error rates, we use the fraction of packets lost when discussing the drop prevalence in this section as opposed to the bit-error rates used in previous sections. All corruption rates used in Section 2 represent less than a 1% packet drop rate. In this section, most of the packet error rates used are at least 1%. In other words, the experiments presented in this section generally have a higher prevalence of corruption than in the experiments presented in Section 2.

The first set of simulations involve a single TCP flow across the network described above. In these simulations corruption-based losses are applied to only the data packets traversing the bottleneck link (i.e., not for the ACK traffic flowing back to the sender). Fig. 8 shows TCP performance as a function of the corruption-rate plotted on a log–log scale. The plot shows the performance drop-off of stock TCP SACK. In addition, the figure illustrates that both versions of CETEN offer better performance than stock TCP SACK—even though CETEN's performance does decrease as the corruption rate increases.

The cause of CETEN's performance reductions at high packet corruption rates is largely dropped retransmissions. TCP SACK relies on the RTO timer to cope with retransmissions that are dropped. The RTO timer represents a lengthy inactive period, as well as a second *cwnd* reduction. We do note that even though performance is dropping off at a packet corruption rate of 20%, CETEN_A still achieves more than an order of magnitude increase when compared to stock TCP.

Another notable aspect of Fig. 8 is the difference in performance between CETEN_A and CETEN_P—even though they are intuitively attempting to achieve the same notion. The notion behind CETEN_P is that it reduces *cwnd* roughly *the right number of times* over the course of a long transfer to compensate for network congestion. However, on any specific loss event CETEN_P could "guess wrong" and take the "wrong" action. For instance, if a loss is caused by corruption, CETEN_P may decide to reduce *cwnd*. The hope is that later when there is a congestion-based loss CETEN_P will even things out by not reducing *cwnd*. However, in the simulations presented in Fig. 8 this notion does not play out as planned. When there is only one connection in the network that connection is solely responsible for network congestion. Therefore, when congestion occurs and CETEN_P decides not to reduce *cwnd*, the congestion is still present and more losses will occur. In effect, CETEN_P is *forced to reduce* the *cwnd* when congestion occurs. So, while CETEN_P prevents the *cwnd* from being reduced in some cases when corruption occurs, the connection does not get the entire benefit envisioned, and hence, experiences lower performance compared to CETEN_A. In a network with more statistical multiplexing CETEN_P may perform better (closer to CETEN_A) because a single connection will not be the sole cause of congestion. Therefore, when a congestion event occurs and a CETEN_P connection maintains its *cwnd* the connection may not incur further congestion because competing traffic will also likely be backing off.

The second set of CETEN simulations involves competing traffic. The four node topology described above is again employed. In this set of tests we run a single TCP connection in each direction across the network. In addition, we run five on/off constant-bit rate (CBR) flows across the network in each direction. The CBR flows are driven by an exponential random process that has a mean on time of 2.5 s and a mean off time of 10 s. When on, each CBR flow sends at 1 Mbps. Therefore, when all the CBR flows are running they would consume the bottleneck capacity. The TCP connection is set up as described for the single flow tests above. Corruption-based losses are inserted in both directions of the bottleneck link according to a uniform random process.

Fig. 9 shows TCP performance as a function of the corruption rate applied to the bottleneck link on a log–log plot. Again, this plot illustrates the power of CETEN to increase performance over

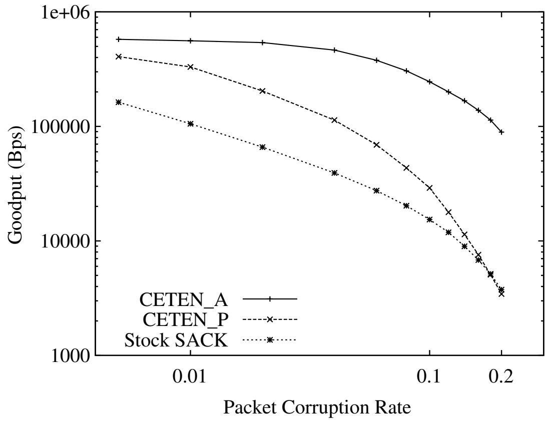


Fig. 8. CETEN_P and CETEN_A vs. stock SACK TCP.

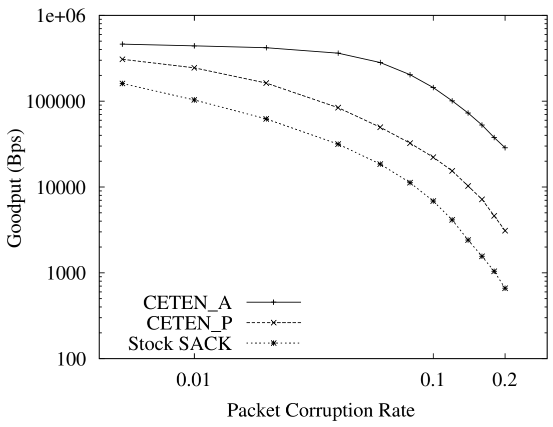


Fig. 9. CETEN$_P$ and CETEN$_A$ vs. stock SACK TCP with competing traffic.

stock TCP. Also, this plot shows that CETEN$_P$ provides better performance enhancement at high error rates than shown above for single flow experiments. This suggests that the above note about CETEN$_P$ working better in an environment with a high degree of statistical multiplexing may be accurate (but must be verified completely using more complex simulations with competing congestion-aware traffic). With competing traffic CETEN shows performance improvements of 1–2 orders of magnitude over stock TCP SACK at high error rates.

4.5. Discussion

Our preliminary simulations have shown CETEN to be a promising approach in mitigating the problems corruption-based losses pose to TCP performance in wireless and satellite networks. However, many questions remain before the community could even consider CETEN for wide-scale adoption, such as: How do we derive corruption-survival probabilities? Over what time scale? Should TCP and/or the router keep a running or smoothed average of the congestion and corruption rates? How burdensome is CETEN on the forwarding engines in routers? What can or should be done about receivers sending bogus corruption reports in an attempt to game congestion control [42]? These and many additional questions will be the subject of future work in this area.

5. Summary

In this paper, we consider the problem of enhancing TCP performance in the face of corruption losses, and make the following contributions. We confirm previous studies that show corruption-based loss causes performance problems for TCP. In addition, we illustrate a plausible upper bound on the performance TCP could attain with perfect knowledge about the causes of loss. We present a detailed taxonomy of the space of mitigations for the issues caused by corruption-based loss. This taxonomy is a useful road-map to researchers who wish to pursue alternative mitigation approaches. Finally, we offer a preliminary illustration of the potential benefits of a previously unexplored portion of the mitigation space. We show that CETEN is a promising technique for improving TCP performance in environments with non-negligible corruption-based packet loss. While promising, CETEN also has numerous theoretical and practical issues that require attention before the strategy will be useful for general, wide-scale deployment.

Acknowledgments

The work presented in this paper was conducted with funding from NASA's Glenn Research Center and NASA's Earth Science Technology Office under contract number NAS3-99175, as well as from the National Science Foundation under grant number 0205519. James Sterbenz and Mark Allman performed a portion of this work while they were with BBN Technologies. Will Ivancic (NASA GRC), Steve Polit (BBN Technologies) and the anonymous reviewers provided very helpful comments on an earlier version of this manuscript. Shawn Ostermann (Ohio University) and David Mankins (BBN Technologies) contributed to the CETEN experiments outlined in Section 4.

References

[1] I.F. Akyildiz, G. Morabito, S. Palazzo, TCP-Peach: a new congestion control scheme for satellite IP networks, IEEE/ACM Transactions on Networking 9 (3) (2001) 307–321.

[2] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, Early retransmit for TCP and SCTP, Internet-Draft draft-allman-tcp-early-rexmt-03.txt, December 2003, work in progress.

[3] M. Allman, W. Eddy, S. Ostermann, Estimating loss rates with TCP, ACM Performance Evaluation Review 31 (3) (2003).

[4] M. Allman, V. Paxson, W. Stevens, TCP congestion control, Request for Comments 2581, April 1999.

[5] ATM Forum, ATM User Network Interface (UNI) Signalling Specification Version 4.1, af-sig-0061.002, April 2002, available from www.atmforum.com/standards/approved.html.

[6] F. Baker (Ed.), Requirements for IP Version 4 Routers, Request for Comments 1812, June 1995.

[7] A. Bakre, B.R. Badrinath, I-TCP: indirect TCP for mobile hosts, in: Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS), May 1995.

[8] H. Balakrishnan, R.H. Katz, Explicit loss notification and wireless web performance, in: Proceedings of the IEEE Globecom Internet Mini-Conference, Sydney, Australia, November 1998.

[9] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Transactions on Networking 5 (6) (1997) 756–769.

[10] S. Biaz, N.H. Vaidya, Distinguishing congestion losses from wireless transmission losses: a negative result, in: Proceedings of the Seventh International Conference on Computer Communications and Networks (IC3N), New Orleans, October 1998.

[11] E. Blanton, M. Allman, K. Fall, Lili Wang, A conservative selective acknowledgment (SACK)-based loss recovery algorithm for TCP, Request for Comments 3517, April 2003.

[12] J. Border, M. Kojo, J. Griner, G. Montenegro, Z. Shelby, Performance enhancing proxies intended to mitigate link-related degradations, Request for Comments 3135, June 2001.

[13] L. Brakmo, S. O'Malley, L. Peterson, TCP Vegas: new techniques for congestion detection and avoidance, in: Proceedings of ACM SIGCOMM, August 1994.

[14] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, R. Wang, TCP Westwood: end-to-end congestion control for wired/wireless networks, Wireless Networks 8 (2002) 467–479.

[15] D.W. Davies, The control of congestion in packet-switching networks, IEEE Transactions on Communications COM-20 (3) (1972) 546–550.

[16] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, N. Vaidya, End-to-end performance implications of links with errors, Request for Comments 3155, August 2001.

[17] W. Eddy, Improving TCP performance with path error rate information, Master's Thesis, Ohio University, March 2004.

[18] W. Eddy, S. Ostermann, M. Allman, New techniques for making transport protocols robust to corruption-based loss (submitted for publication).

[19] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe, Reno, and SACK TCP, ACM Computer Communication Review 26 (3) (1996) 5–21.

[20] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the internet, IEEE/ACM Transactions on Networking 7 (4) (1999) 458–472.

[21] S. Floyd, TCP and explicit congestion notification, ACM Computer Communication Review 24 (5) (1994) 10–23.

[22] C. Hayes, Analyzing the performance of new tcp extensions over satellite links, Master's Thesis, Ohio University, August 1997.

[23] V. Jacobson, Congestion avoidance and control, in: Proceedings of ACM SIGCOMM '88, Stanford, CA, USA, August 1988.

[24] M.A. Jolfaei, B. Heinrichs, M.R. Nazeman, Improved TCP error control for heterogeneous WANs, in: Proceedings of the IEEE National Telesystems Conference, San Diego, CA, USA, May 1994.

[25] D. Katabi, M. Handley, C. Rohrs, Internet congestion control for future high bandwidth-delay product environments, in: Proceedings of ACM SIGCOMM, August 2002.

[26] S. Kent, R. Atkinson, Security architecture for the internet protocol, Request for Comments 2401, November 1998.

[27] R. Krishnan, M. Allman, C. Partridge, J.P.G. Sterbenz, Explicit transport error notification (ETEN) for error-prone wireless and satellite networks, Technical Report TR-8333, BBN Technologies, March 2002.

[28] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP selective acknowledgment options, Request for Comments 2018, October 1996.

[29] A.J. McAuley, Reliable broadband communication using a burst erasure code, in: Proceedings of ACM SIGCOMM '90, Philadelphia, PA, USA, September 1990.

[30] M. Mellia, M. Meo, C. Casetti, TCP smart framing: a segmentation algorithm to improve TCP performance, in: Proceedings of the 2nd International Workshop on QoS in Multiservice IP Networks (QoS-IP 2003), February 2003.

[31] ns-2 simulator, available from http://www.isi.edu/nsnam/ns/index.html.

[32] V. Paxson, J. Mahdavi, A. Adams, M. Mathis, An architecture for large-scale internet measurement, IEEE Communications 36 (8) (1998) 48–54.

[33] V. Paxson, M. Allman, Computing TCP's retransmission timer, Request for Comments 2988, November 2000.

[34] J. Postel, Internet control message protocol, Request for Comments 792, September 1981.

[35] J. Postel (Ed.), Transmission control protocol, Request for Comments 793, September 1981.

[36] K. Ramakrishnan, S. Floyd, A proposal to add explicit congestion notification (ECN) to IP, Request for Comments 2481, January 1999.

[37] K. Ramakrishnan, S. Floyd, D. Black, The addition of explicit congestion notification (ECN) to IP, Request for Comments 3168, September 2001.

[38] K.K. Ramakrishnan, R. Jain, A binary feedback scheme for congestion avoidance, ACM Transactions on Computer Systems 8 (2) (1990) 158–181.

[39] L. Rizzo, Effective erasure codes for reliable computer communication protocols, ACM Computer Communication Review 27 (2) (1997) 24–36.

[40] N. Samaraweera, Non-congestion packet loss detection for TCP error recovery using wireless links, IEE Proceedings in Communications 146 (4) (1999) 222–230.

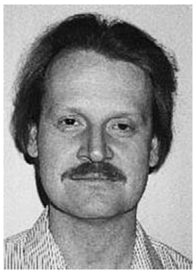
[41] N. Samaraweera, G. Fairhurst, Explicit loss indication and accurate RTO estimation for TCP error recovery using satellite links, IEE Proceedings in Communications 144 (1) (1997) 47–53.

[42] S. Savage, N. Cardwell, D. Wetherall, T. Anderson, TCP congestion control with a misbehaving receiver, ACM Computer Communications Review 29 (5) (1999) 71–78.

[43] N. Shacham, P. McKenny, Packet recovery in high-speed networks using coding and buffer management, in: Proceedings of IEEE INFOCOM '90, San Francisco, CA, June 1990.

[44] J. Stone, C. Partridge, When the CRC and TCP checksum disagree, in: Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August 28–September 1, 2000.

Rajesh Krishnan is a Senior Scientist at BBN Technologies, Cambridge, MA, USA. At BBN's Internetwork Research Department since 1997, he has led and contributed to several research efforts in the field of networking. He holds the Ph.D. (2004) and M.S. (1996) degrees in Computer Engineering from Boston University, Boston, MA, USA, and the B.E. (1991) degree with Honours in Electrical Engineering from the Regional Engineering College, Durgapur, West Bengal, India. From 1991–1994, he worked for the Tata Engineering and Locomotive Company Limited, Jamshedpur, Bihar, India. He is a member of the ACM, the IEEE, and the IEEE Communications Society.



James P.G. Sterbenz is a Visiting Research Scientist in the Computer Networks Research Group at the University of Massachusetts, Amherst. He has been PI for several DARPA and NASA funded research programs in the areas of survivable, disruption-tolerant, mobile, wireless, and active networking, and TCP and web performance. He has previously held senior research staff and management positions at BBN Technologies, GTE Laboratories, and IBM, and holds a D.Sc. in Computer Science from Washington University in St. Louis. He is program co-chair for IEEE Hot Interconnects 2004, and was program co-chair of IWAN 2003, 2002, and PfHSN'99. He is past chair of the IEEE Communications Society Technical Committee on Gigabit Networking, chair of the IFIP Protocols for High Speed Networks Steering Committee, member of the IFIP Active Networks steering committee, senior member of the IEEE, member of the ACM, IEE (UK), IEICE (Japan), the Internet Society Interplanetary Special Interest Group, and on the editorial board of IEEE Network. He is the author of the book 'High-Speed Networking: A Systematic Approach to High-Bandwidth Low-Latency Communication'.

Wesley M. Eddy is currently a researcher at NASA's Glenn Research Center. The work described here was completed while he was a student at Ohio University, where he earned the B.S. and M.S. degrees in computer science in 2002 and 2004. His research interests include protocols for mobile hosts and extending transport protocols for NASA missions.

Craig Partridge is a Chief Scientist at BBN Technologies, where he has done data communications research for the past 20 years. He is best known for designing how Internet email is routed, and for his work on high performance networking. A Fellow of the IEEE and the ACM, he received his A.B., M.Sc. and Ph.D. degrees from Harvard University.

Mark Allman is a Computer Scientist with the International Computer Science Institute. His current research interests are in the areas of transport protocols, congestion control, measuring network dynamics and network security. He is involved in the Internet Engineering Task Force, where he has chaired several working groups and BoFs and is currently a member of the Transport Area Directorate. He also chairs the Internet Measurement Research Group within the Internet Research Task Force. He has served on the program committee for numerous conferences and workshops. He holds B.S. and M.S. degrees in computer science from Ohio University and is a member of the ACM.