# New Techniques for Making Transport Protocols Robust to Corruption-Based Loss


Wesley M. Eddy
NASA GRC / Verizon
weddy@grc.nasa.gov

Shawn Ostermann
Ohio University
ostermann@eecs.ohiou.edu

Mark Allman
ICSI Center for Internet Research
mallman@icir.org



## ABSTRACT

Current congestion control algorithms treat packet loss as an indication of network congestion, under the assumption that most losses are caused by router queues overflowing. In response to losses (congestion), a sender reduces its sending rate in an effort to reduce contention for shared network resources. In network paths where a non-negligible portion of loss is caused by packet corruption, performance can suffer due to needless reductions of the sending rate (in response to "perceived congestion" that is not really happening). This paper explores a technique, called Cumulative Explicit Transport Error Notification (CETEN), that uses information provided by the network to bring the transport's long-term average sending rate closer to that dictated by only congestion-based losses. We discuss several ways that information about the cumulative rates of packet loss due to congestion and corruption might be obtained from the network or through fairly generic transport layer instrumentation. We then explore two ways to use this information to develop a more appropriate congestion control response (CETEN). The work in this paper is done in terms of TCP. Since numerous transport protocols use TCP-like congestion control schemes, the CETEN techniques we present are applicable to other transports as well. In this paper, we present early simulation results that show CETEN to be a promising technique. In addition, this paper discusses a number of practical and thorny implementation issues associated with CETEN.


## 1. INTRODUCTION

This paper describes a technique for coping with the potential suboptimal performance caused by the transport layer's inability to derive the reason a packet is dropped[1]. Today's transport protocols assume that packet loss is the result of a queue in the network overflowing due to congestion [15]. However, this assumption is false when links in the network path corrupt non-negligible percentages of packets (which are subsequently dropped to ensure reliability). In the case of purely corruption-based loss, the transport would ideally continue transmitting (including retransmitting data lost due to corruption in the case of a reliable transport) without altering its sending rate. Assume the packet loss rate on a path is $p$, which is the sum of the loss rate due to congestion, $c$, and the loss rate due to corrupted packets[2], $e$, so that $p = c + e$. [25] shows that TCP throughput is proportional to $\frac{1}{\sqrt{p}}$. The goal of the work outlined in this paper is to make TCP's throughput proportional to $\frac{1}{\sqrt{c}}$ (i.e., only the portion of the total loss rate caused by congestion). This approach should yield TCP-friendliness in the face of network congestion while increasing TCP's performance in the face of corruption-based loss.

Several approaches have been proposed and investigated in the literature to mitigate the problems caused by corruption-based loss, as follows.

**Fix the errors at the link layer.** This mitigation method calls for links with known high corruption probabilities to repair the losses locally using Forward Error Correction (FEC) [2] or Automatic Repeat reQuest (ARQ) [32]. The cost of such repair may be a loss of available bandwidth or the introduction of new path dynamics (extra delay, jitter, packet reordering, etc.).

**Split the transport connection.** These techniques call for intelligent entities at the end-points of a link with known high corruption rates to silently terminate and re-initiate transport layer connections to hide losses on the error-prone link from the endpoints. I-TCP [3] is an example of the strategy. Consider the case of a connection between a sender, $S$, and a receiver, $R$, that traverses a link between two intermediate hosts, $I_1$ and $I_2$. One variant of this approach calls for three different transport layer connections to be established (transparently to the user): between $S$ and $I_1$, between $I_1$ and $I_2$ and between $I_2$ and $R$. This effectively isolates the corruption based losses to the connection between $I_1$ and $I_2$, which can be tuned to deal with such losses in a more aggressive manner than TCP's standard response. This class of mitigation may not work when packet headers are encrypted (e.g., when using IPsec [20]).

**Corruption notifications.** This class of mechanisms calls for sending messages to the source or destination addresses of packets that are found to be in error by an intermediate host. These messages can be sent in-band (as in [5, 4]) on the next packet that arrives from the connection in question or using an out-of-band signal such as an ICMP [26] message. The message serves to indicate to the endpoint that a corruption-based loss occurred with the implication being that the host could continue sending without adjusting its congestion control state. The downsides of this strategy are ($i$) that the messages are based on potentially-corrupted information and thus could be sent to an endpoint not involved in the connection and ($ii$) in the case of out-of-band messages, an additional bandwidth

[1] The discussions in this paper are in terms of TCP [27], but also apply to alternate transport protocols and congestion control schemes that use packet loss as a signal of network congestion (e.g., SCTP [33] and TFRC [11, 14]).

[2] Note: There are situations whereby a packet loss may, in fact, indicate both congestion and corruption. For instance, a packet could be marked as experiencing congestion via ECN [28] and then later be corrupted and dropped. Also, in some wireless networks, contention for the channel (congestion) may ultimately cause packet corruption. While these cases certainly deserve careful thought, we defer them to future work and do not consider them further in this paper.

requirement is imposed.

Hybrid schemes that intermix the above strategies have also been developed, such as TCP *snoop* [6]. TCP *snoop* does not split the end-to-end TCP connection, but rather provides transport layer loss repair at the ends of a link that is known to corrupt packets, while also delaying the loss signal from reaching the TCP endpoint. TCP *snoop* retains the notion that the endpoint is ultimately responsible for loss recovery, while attempting local loss recovery to obviate the need for end-to-end recovery of losses due to corrupted packets (which also prevents a needless congestion response).

In this paper we offer an initial study of an additional approach for mitigating the impact of corruption-based losses called Cumulative Explicit Transport Error Notification (CETEN). We share the name CETEN with an earlier work [21] that developed many of the concepts. We refer to the specific scheme used in the original work as $CETEN_O$ in this paper. The key idea behind CETEN is that the mechanism does not attempt to derive the reason for specific packet losses, but rather uses aggregate information provided by the network in an attempt to ensure that TCP's *long-term average sending rate* is appropriate for the *congestion level* of the current network. The network provides information about the corruption-based loss rate, $e$. The TCP sender uses this information along with an estimate of the total loss rate, $p$, to determine the fraction of losses due to corruption, $\frac{e}{p}$ (and, therefore, the fraction due to congestion, $\frac{c}{p} = \frac{p-e}{p}$). TCP's congestion response is then altered to only take into account the congestion-based losses when determining the sending rate.

The remainder of this paper is organized as follows. § 2 discusses methods for gathering enough information to estimate both the total loss rate and its components. In § 3 we use the information gathered via the mechanisms sketched in § 2 to refine TCP's congestion response to more accurately reflect only the congestion-based losses. § 4 presents the results of preliminary simulations of the CETEN techniques. § 5 discusses several implementation issues. Finally, § 6 gives our conclusions and discusses future work in this area.

## 2. GATHERING INFORMATION

The first problem we tackle is gathering all the information required to untangle the loss story into its component causes. For a static path through the network, total packet loss rate, $p$, is the sum of the packet loss rate due to congestion, $c$, and the packet loss rate due to corruption (or errors), $e$. Once obtained, this information might be used by a TCP sender to adjust its congestion control response to include only congestion related loss. Unfortunately, the TCP sender has none of the needed information on hand. The remainder of this section discusses various potential methods of obtaining enough information to drive a better congestion control response.

### 2.1 Gathering the Corruption Loss Rate

As noted in the previous sections the TCP endpoints have no information about *why* packet losses occur. Therefore, to figure out the loss rate caused by corruption, $e$, the intermediate nodes along a path need to be actively involved. While engaging the intermediate nodes has the potential to provide a rich set of information it is also problematic in terms of both performance and deployment. These practical issues are discussed further in § 5. In the following sections we provide a taxonomy of possible schemes for endpoints and intermediate nodes along the network path to interact to provide an estimate of $e$ to the TCP sender.

#### *2.1.1 Out-of-Band Router Queries*

This method calls for the TCP sender to issue some form of ICMP [26] query to each router (via TTL limiting, ala *traceroute*) along the path requesting the corruption loss rate on the attached link. Current routers generally keep the counters necessary to calculate the fraction of corrupted packets that arrive. The advantages of this approach are a rich set of information about the network path and no on-the-wire changes for the network or transport protocols (i.e., for the packets of the data transfer). Changes at the transport layer are particularly burdensome since every transport protocol would have to be modified. Disadvantages of ICMP-based router queries are that $(i)$ additional traffic is required to collect the information, $(ii)$ ICMP messages are unreliable, which will add complexity to the endhost to robustly obtain the path's corruption information and $(iii)$ ICMP messages can be forged, leading to a sender having a bogus understanding of the network properties.

#### *2.1.2 Router Advertisement*

A closely related technique to the above ICMP request/reply scheme would be a router advertisement scheme similar in spirit to several IP traceback proposals (e.g., [31]). Under such a scheme a router would simply choose every $m$-th packet and send a corruption estimate for its link to the sender of the chosen packet. The advantages of this scheme are similar to those outlined for the ICMP request/reply mechanism outlined above. That is, the sender gets a rich set of information about the corruption rate without changing the on-the-wire network or transport protocols. This scheme cuts down on the bandwidth required when compared with the ICMP request/response scheme sketched above due to the implicit request. However, it still requires extra packets to be formed and transmitted by the intermediate nodes. The resources are further controlled by giving the routers an explicit knob to control the overhead (i.e., $m$). This scheme exacerbates the reliability concerns discussed in the last section. When a sender is making an explicit request the sender can reasonably judge when to send another request in the absence of a response. However, if a router advertisement is lost the host will have to wait for $m$ packets to traverse that router before having another chance at obtaining the corruption information for the given link. Finally, router advertisements can also suffer from the ICMP spoofing issue outlined above.

#### *2.1.3 Out-of-Band Cumulative Queries*

Another technique that can be used to gather corruption rates from intermediate nodes is out-of-band *cumulative* queries. This scheme is based on an in-band version originally outlined in [21] (which will be discussed below). This mechanism calls for an ICMP message to be transmitted from the TCP sender's host to the TCP receiver's host that encodes the "corruption survival probability". The corruption survival probability is $1 - e$, or the probability that a packet will traverse the entire network path from sender to receiver without being corrupted. The sender initializes the probability to 1.0. Each hop along the path multiplies the probability contained in the ICMP packet with the corruption survival probability of the hop's incoming link. The result then replaces the probability in the ICMP message before the packet is forwarded. The receiving host then echos corruption reports to the sender in another ICMP message. The key difference between this scheme and the mechanisms given in the last sections is that the cumulative query pushes the aggregation of path information to the intermediate nodes themselves rather than requiring the TCP sender to aggregate the information from multiple queries/advertisements. While the out-of-band nature of the scheme still requires additional traffic, the cumulative notion lessens this traffic since a single probe characterizes the entire path, as opposed to characterizing a link at a time, as outlined in the schemes above. The spoofing and reliability concerns discussed above also apply to this mechanism.

An additional point about this cumulative out-of-band mechanism is that all the routers on a path are not required to support these queries. A router connected to links where corruption is negligible will add nothing to the estimate of the corruption rate. That is, the router will essentially multiply the value from the ICMP message by 1.0 and re-encode the ICMP message. Therefore, such routers would not need to waste resources participating in the effort to gather $e$.

Finally, we note that [19] discusses similar use of ICMP probing in the context of determining a number of different path properties (minimum MTU, minimum bandwidth, maximum delay, etc.). [19] mentions that one of the properties that could be collected using ICMP messages is the "maximum error rate", which would exactly match the cumulative scheme we outline here in the case of a single corruption-prone link in the path.

#### *2.1.4 In-Band Requests*

Another option would be requests attached to the data packets in the transfer. For instance, an IP option could include a TTL$'$ field and an empty field for the corruption rate. The sender would specify a TTL$'$ in the option. When a router along the path decremented the IP TTL to the value of TTL$'$ then the router would insert the probability of corruption-based loss into the empty field in the IP option. The receiver of the packet could then echo the received probability back to the sender in ACKs. The sender would have to iterate through a range of TTLs to determine the corruption rate of the entire path.

This in-band scheme has several advantageous properties. First, no additional packets are required to be formed and transmitted in the network, since the reports are piggybacked on TCP segments, saving network resources. While routers still have to do work to process the in-band requests the work is not likely as intensive as forming up a new packet. The reports are more reliable than the out-of-band schemes discussed above. Reports on data segments are reliable in that if the data packet is lost the retransmit can be used to request the information again. The reports that are echoed on ACKs are not reliable since ACKs can be lost with little consequence in TCP. However, sending reports on multiple ACKs will increase the chances of the TCP sender getting the information (similar to the way selective acknowledgments [22] and ECN [28] congestion indications are sent on multiple ACKs for robustness purposes). Finally, routers that are not connected to corruption-prone links do not need to participate in this scheme (since such routers would be sending a "no corruption loss" report anyway).

This in-band scheme has disadvantages, as well. With the out-of-band schemes the intermediate node can easily find the requests for corruption information in the traffic stream (due to the ICMP packet types). When using the in-band scheme *any* packet could contain a request for corruption information and therefore participating intermediate nodes will have to examine *all* packets. The in-band mechanism also suffers from concerns over forged reports. However, these concerns are somewhat mitigated because an attacker would have to form an acknowledgment that is consistent with a given TCP connection. This reduces the chances of a blind attack against the TCP connection when compared to blind attacks using one of the out-of-band scheme presented above.

#### *2.1.5 In-Band Cumulative Queries*

A final class of strategies for gathering corruption information from the intermediate nodes is to use in-band cumulative queries. This scheme combines the ideas from § 2.1.3 and § 2.1.4. The information gathered is cumulative, but rather than using ICMP messages, the corruption survival probability is encoded in a field or option of the data packets and returned in a field or option in ACK packets. This notion was used in [21]. The pros and cons of this scheme are as discussed above in § 2.1.3 and § 2.1.4

## 2.2 Estimating the Total Loss Rate

At first glance it appears as though TCP would have an accurate estimate of the total loss rate, $p$, since TCP is a reliable protocol that retransmits segments that are lost (for whatever reason). However, TCP is often too aggressive in its loss recovery behavior which leads to unnecessary retransmissions. [1] presents Internet measurements that show a simple count of retransmissions provides only a gross estimate of the loss rate. For instance, [1] shows that TCP Reno's retransmission rate is more than 10% higher than the actual loss rate in two-thirds of the transfers studied and in 16% of the transfers more than twice as many retransmits as are required are sent. The results for TCP SACK are better, showing a median difference of 2% between the retransmission rate and the loss rate and 75% of the connections spuriously retransmitting no more than 10% of the time. Given the inaccuracy of using the retransmission count as $p$ we explore several methods for obtaining more accurate information in the following sections.

#### *2.2.1 Sender-Side Estimation*

One approach to obtain a better estimate of $p$ is to have the sender infer the loss rate via the retransmission count, as well as additional information that might be available. For instance, TCP's DSACK option [12] calls for the receiver to report duplicate segment arrivals to the sender. The sender could use this information in conjunction with the retransmission count to calculate a more accurate estimate of the loss rate[3]. Even without DSACK, TCP can use hints from the ACK stream to infer that a segment has been received more than once. For instance, since a needless retransmit does not update any state at the receiver a duplicate ACK (or an ACK with no new SACK information) is transmitted by the receiver. This case is more complicated than the DSACK case because a duplicate ACK can be sent for several reasons and therefore the TCP sender must attempt to derive the cause of the duplicate ACK.

[1] provides the details of a sender-side loss estimation scheme (known as $LEAST$), as sketched above. In addition, [1] presents Internet measurements showing that estimating the loss rate within 10% of the true loss rate is possible in over 90% of the transfers (regardless of TCP variant employed). Furthermore, when using DSACK a TCP sender is able to estimate within 1% of the true loss rate in over 97% of the transfers and within 10% of the true loss rate in over 99% of the transfers.

In the context of CETEN, accurately estimating the total loss rate from TCP sender with no additional protocol machinery (ala $LEAST$) is ideal. The outstanding question with using sender-side estimation is whether the accuracy is "good enough".

#### *2.2.2 Receiver-Side Estimation*

An alternate to sender-side estimation is for the receiver to estimate the loss rate and transmit the information back to the sender in ACK packets. This is analogous to the approach used in TFRC [11, 14] for rate-based congestion control. Such a scheme could

[3]DSACKs are not sent reliably and therefore an exact determination of the loss rate is not always possible due to ACK loss in the network. In addition, spurious retransmissions are not the only cause of DSACKs. Packet duplication in some network element could also cause such behavior, for instance.

disregard spurious retransmits in the loss estimate. On the other hand, simply counting segments that arrive out of order may cause an overestimate of the loss rate due to packet reordering in the network [7]. Finally, in the CETEN context a receiver-based loss estimator would need a new TCP option to convey the loss rate to the sender for use in adjusting the congestion response.

We are not aware of any concrete scheme for accurately estimating the loss rate from the TCP receiver's vantage point. Future work could include designing such a scheme and comparing the accuracy with that of $LEAST$. In addition, such a scheme would be of general use in network monitoring – even if it was not used for CETEN.

#### 2.2.3 *Endpoint Cooperation*

A third class of methods to gather the total loss rate is to use a cooperative counting mechanism between the TCP sender and receiver. Such a mechanism would call for the sender to keep track of the total number of segments sent, $S_s$, within a particular sequence range. The sender would then query the receiver for the total number of segments that arrived, $S_a$, in the given sequence range. The number of losses in the given range is then easily determined as $S_s - S_a$. The advantage of this scheme lies in its simplicity and robustness.

One disadvantage of this scheme is that additional state is required at both the sender and receiver – although, we expect that a simple table of counters does not represent a high barrier to using such a scheme. Another cost of this method is that additional on-the-wire protocol machinery is required to exchange information about the number of segments that arrive at the receiver.

#### 2.2.4 *Querying Routers*

A final class of schemes to obtain an estimate of $p$ would be to query the routers along a given network path as discussed above in terms of gathering $e$. To gather the total loss rate a scheme could either gather $p$ directly or gather $c$ and calculate $p$ as $c + e$. The various classes of mechanisms discussed in § 2.1 (in the context of determining $e$) with their associated costs and benefits would apply to gathering congestion information or total loss rates, as well.

[21] discusses a particular cumulative in-band version by gathering $c$ from the network in the context of CETEN. The notion is similar to that discussed in § 2.1.5 in that a new header field is introduced that represents the congestion survival probability $(1 - c)$, which is initialized to 1.0 by the TCP sender. Each hop updates the value in the packet based on their level of congestion. The value is echoed back to the sender in ACK packets.

One practical disadvantage of counting on help from the routers to determine $p$ or $c$ is that *all the routers must participate* or the estimates will end up low and the congestion response will not be accurate. As discussed above ubiquitous deployment is not required when querying routers for corruption information only. In practical terms, ubiquitous deployment is a significant disadvantage to relying on help from intermediate nodes to determine $p$ or $c$.

## 2.3 Practical Concerns

A number of the schemes sketched in the previous two sections query the intermediate nodes in the network for information — either via header options or ICMP messages. Above we have discussed the idealistic tradeoffs in different gathering techniques (e.g., additional bandwidth requirements vs. ease of finding information requests). However, there are another set of tradeoffs that also come into play if these techniques were to be used in real networks. The Internet has evolved away from its textbook description in a number of areas for many reasons. A case in point is that middleboxes can cause unexpected behavior. [23] outlines experiments to over 80,000 web servers that provide two key findings that impact the information gathering techniques presented above.

- Roughly 17% of the web servers tested do not support Path MTU Discovery (PMTUD) [24] due to ICMP packets being discarded by an intermediate node. (30% of the web servers did not attempt PMTUD and so, 17% is a lower bound.)
- When an undefined IP option (simulating a freshly minted IP option) is included on the SYN segment sent by the client, 70% of the connection initiations fail, compared with 2% when no IP options are included.

These two findings will ultimately impact on the way protocols and protocol extensions are designed. And, in particular, if the techniques outlined in this paper are ever to deployed these issues will have to be tackled.

## 2.4 Choosing Gathering Techniques

The $CETEN_O$ scheme [21] gathered estimates of both $c$ and $e$ via cumulative in-band querying of the routers in a particular path. These values were computed by routers using a fast moving average.

The work presented in this paper gathers $e$ using the cumulative in-band querying technique (from § 2.1.5). We made this choice mainly due to the low overhead and ease of implementation. We do not necessarily argue that this technique would be the best path forward for real implementations. Our goal in this paper is to gain an initial understanding of how CETEN techniques may work and not to do fine-grained engineering of protocol extensions. For gathering $p$ we implement the $LEAST$ algorithms in the TCP sender (as discussed in § 2.2.1). We consider this to be more practical for CETEN than to require assistance from *all* the routers along a path for CETEN to work properly. In addition, using $LEAST$ is more palatable from a deployment standpoint than the receiver-based scheme or the cooperative counting technique due to the communication requirements these solutions impose. However, future work could include comparing CETEN when using total loss information gathered in different ways to gauge how sensitive CETEN is to the accuracy of the $p$ estimates. In contrast to $CETEN_O$, because our estimate of $p$ is a long-term average, we also use long-term averages of $e$ which has the advantage of easing some computational burden on routers as well.

# 3. A NEW CONGESTION RESPONSE

As discussed above, TCP performance has been shown to be proportional to $\frac{1}{\sqrt{p}}$, where $p$ is the total packet loss rate. However, by gathering information about the components of $p$, as discussed in the last section, our goal is to change TCP's congestion control response to be proportional to $\frac{1}{\sqrt{c}}$, where $c$ is the congestion-based loss rate. As further motivation, figure 1 is a log-log plot that shows the performance predicted by the TCP model [25] as a function of $p$. The network is assumed to have a round-trip time (RTT) of 0.5 seconds and the modeled TCP has a maximum segment size (MSS) of 1460 bytes. The performance is given for both a stock TCP and an alternate TCP that is able to determine that 75% of the packet loss is caused by corruption (i.e., $\frac{e}{p} = 0.75$). As shown in the plot, a TCP that can derive the true congestion rate from the overall loss rate enjoys performance benefits.

While TCP's ultimate performance is well modeled by the TCP equation, the formula is not directly implemented in the protocol. Therefore, simply replacing $p$ with $c$ as in the above idealized

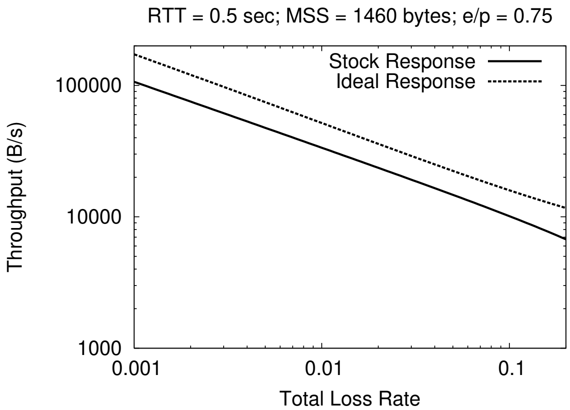



**Figure 1: Performance computed using Padhye's TCP model with and without packet corruption being treated as congestion.**

analysis is not feasible[4]. TCP's sending rate is controlled using a *congestion window* (*cwnd*) which dictates the number of segments[5] that can be transmitted into the network before receiving an acknowledgment. TCP obeys additive-increase, multiplicative-decrease (AIMD) congestion control [15]. In the absence of loss, the TCP sender increases *cwnd* by roughly 1 segment per RTT. When loss is observed the TCP sender halves *cwnd*. These increase/decrease decisions ultimately yield performance that is predicted by the TCP model. The next two subsections explore two different methods for altering TCP's congestion control decisions in an attempt to obtain performance that is consistent with the congestion level in the network rather than the total loss rate.

## 3.1 Probabilistic Response

First we explore a scheme that *probabilistically* performs multiplicative decrease, denoted $CETEN_P$. In this scheme, a coin is weighted with the probability of losing a segment due to congestion, $\frac{c}{p}$. Each time stock TCP would multiplicatively decrease *cwnd*, the coin is flipped. If the flipped coin lands on the "congestion" side, then the TCP sender decreases the *cwnd* by half (i.e., the standard multiplicative decrease). Otherwise, the TCP sender does not change the *cwnd* at all. The notion is that the *long-term average* behavior of the TCP will be correct. That is, the connection will reduce the *cwnd* in roughly the right number of cases over the course of the connection so that the congestion response is approximately proportional to $\frac{1}{\sqrt{c}}$. For instance, assume 10% of the losses on a particular connection are corruption-based. In this case, in roughly 10% of the cases when TCP would normally reduce *cwnd* (and thus its sending rate) by half, *cwnd* is not altered at all, and the sending rate is not reduced. [9] provides an analytic analysis that shows $CETEN_P$ to be proportional to $\frac{1}{\sqrt{c}}$.

## 3.2 Adaptive Response

Next we introduce a CETEN variant that uses *adaptive congestion window reduction*, denoted $CETEN_A$. Rather than probabilistically choosing whether to reduce *cwnd* (by one-half) or not, $CETEN_A$ reduces the *cwnd* on every loss event, but by an amount that may be less than the standard one-half. Specifically, $CETEN_A$ uses the fraction of loss caused by corruption to determine the *multiplicative decrease factor* (MDF) according to the following equation:

$$MDF = \frac{1 + (\frac{e}{np})^k}{2} \tag{1}$$

The $n$ and $k$ parameters allow for the shaping and bounding of the multiplicative decrease factor. Each time a current TCP multiplicatively decreases *cwnd*, the following reduction is used instead:

$$cwnd = cwnd \cdot MDF \tag{2}$$

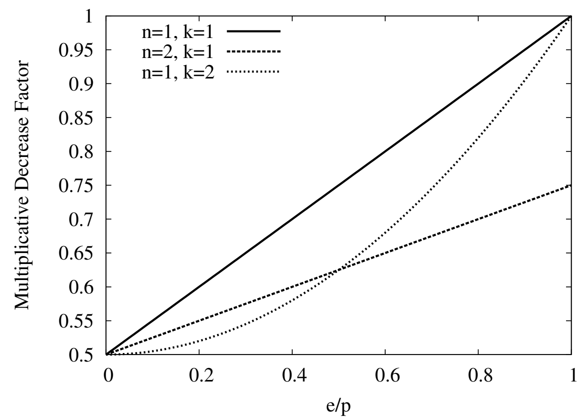



**Figure 2: Multiplicative decrease factor as a function of the fraction of corruption-based losses.**

Figure 2 shows the MDF as a function of $\frac{e}{p}$ for various $n$ and $k$ values. When $n = k = 1$, the MDF exactly counteracts the negative performance impact caused by corruption based loss and provides TCP with the same theoretical performance as the case when only the congestion-based loss is present (this is shown analytically in [9]). When all loss is due to congestion, the standard MDF of $\frac{1}{2}$ is used, whereas when all loss is due to corruption, no decrease in *cwnd* is performed (MDF=1.0).

A simple way to make the MDF more conservative is to increase $n$, as shown in the $n = 2, k = 1$ case in figure 2. In this case the MDF is still linear as the fraction of corruption-based loss increases from 0.0 to 1.0. However, the amount TCP will reduce the window is more than in the $n = k = 1$ case — to the point of *always* requiring a reduction in *cwnd* even when all loss is caused by packet corruption. Such an MDF function will still show suboptimal TCP performance in the face of corruption-based loss, but obtain better performance than standard TCP with an MDF fixed at 0.5. Another method to make the MDF more conservative is to use $k$ to shape the function. For example, the MDF can be shaped as shown in the $n = 1, k = 2$ case on figure 2. In this case, at low corruption rates, $CETEN_A$ is more conservative, but at high corruption rates the MDF grows more rapidly.

Any continuous monotonically-increasing function that is no more aggressive than the $n = k = 1$ line on figure 2 and is based on $\frac{e}{p}$ can be used to determine the MDF. There are tradeoffs no matter what function is considered and an in-depth comparison of MDF functions is beyond the scope of this paper. Therefore, for the remainder of this paper we will use equation 1 with $n = k = 1$ as the MDF.

[4]For rate-based transport protocols, such as TFRC [11, 14], simply replacing $p$ with $c$ is the natural and appropriate approach.

[5]TCP actually tracks *cwnd* in terms of bytes. However, we use segments in the discussion in this paper for simplicity.

# 4. PRELIMINARY SIMULATIONS

This section offers preliminary *ns-2*[6] simulations of $CETEN_P$ and $CETEN_A$. All simulations consist of a sender and receiver separated by two routers. The links between the endpoints and the routers have bandwidths of 1 Gbps and one-way propagation delays of 3 ms. The link between the routers has a bandwidth of 5 Mbps and a one-way propagation delay of 40 ms. The drop-tail queues in the routers can hold 150 packets. The TCPs use SACK [22, 10], DSACK [12], advertised windows of 500 packets, a maximum segment size of 1460 bytes and delayed ACKs with a 100 ms heartbeat timer. The TCP senders use the DSACK variant of the $LEAST$ algorithm to estimate the total loss rate [1]. All simulations run for 1 hour to assess the long-term sending rate of TCP with CETEN. Corruption-based errors are introduced via a uniform random process on the link between the routers through a programmable packet corruption rate. Using a uniform random process for generating corruption based errors is not terribly realistic. However, for this *initial* evaluation we did not want to focus on any particular link technology. That said, Appendix A provides results from simulations involving various bursty loss models; the results are generally consistent with those of the uniform loss process. Therefore, for the results presented in the body of the paper we utilize only a uniform loss model and note that evaluating CETEN under more realistic conditions is future work. The routers in our network do not attempt to dynamically assess the corruption rate, but rather just use the programmed corruption rate. § 5.3 discusses the need for future work in this area. The corruption rate setting is verified to be within 10% of the observed corruption rate in all simulations. Finally, each point on the plots presented in this section represents the mean of 30 random simulation runs.

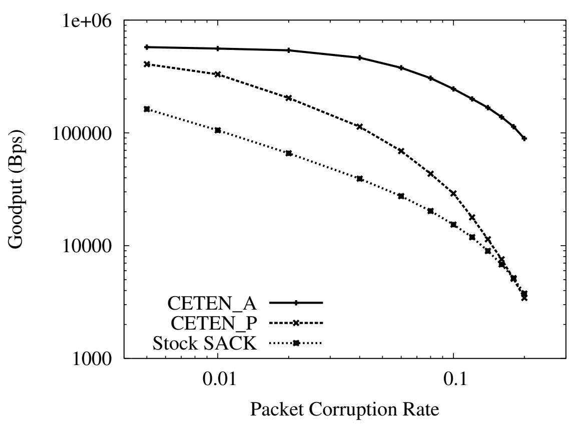



**Figure 3: Performance of a single TCP flow as a function of the corruption rate.**

## 4.1 Single Flow Simulations

Figure 3 shows TCP performance as a function of the corruption rate for a simple simulation scenario consisting of a single TCP flow. Packet corruption is applied only to data packets. The "Stock SACK" line on the plot shows the goodput of unaltered TCP SACK. As expected, the goodput of TCP drops as the number of losses increase (no matter what the cause).

Next we consider the $CETEN_A$ results reported in figure 3. $CETEN_A$ is able to obtain better performance than stock TCP across a breadth of corruption rates. While the goodput obtained by $CETEN_A$ decreases as the corruption rate increases, the goodput obtained by $CETEN_A$ at high corruption rates is still roughly an order of magnitude more than stock TCP SACK. The fundamental cause of the reduction in goodput as the corruption rate increases is lost retransmits which require the expiration of the retransmission timer (RTO) in order to repair when using *ns*' *sack1* TCP variant (as outlined in [10]). The firing of the RTO timer effects goodput in three ways. First, the idle time spent waiting for the RTO to fire represents missed opportunities to transmit data. Second, during this idle period TCP is wasting opportunities to increase *cwnd*. Finally, when the RTO fires the TCP sender clears its "scoreboard" of collected SACK information per RFC 2018 [22]. Therefore, the TCP sender transmits a burst of packets (allowed because $e/p$ is close to 1.0, meaning little *cwnd* reduction occurs) starting at the current cumulative ACK point regardless of whether all of these packets require retransmission. The segments that are unnecessarily retransmitted represent wasted bandwidth that could have been put to better use. The probability of losing a given packet and its retransmission is $O(p^2)$, which explains why the goodput reduction increases as the corruption-rate increases.

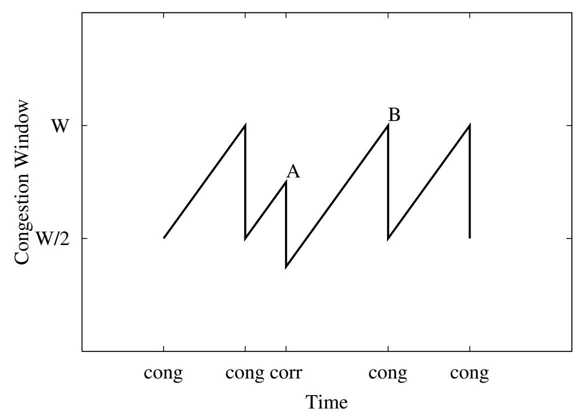



**Figure 4: Example evolution of *cwnd* over time (with losses noted as tick marks on the $x$-axis).**

The final line on figure 3 shows the goodput of the $CETEN_P$ variant. As shown, $CETEN_P$ performs better than stock TCP SACK, but not as well as $CETEN_A$. The cause of the discrepancy in performance between $CETEN_P$ and $CETEN_A$ in these simple simulations is $CETEN_P$'s lack of backoff in certain circumstances. To illustrate this, figure 4 shows the evolution of the *cwnd* over time for a standard TCP connection (without CETEN). The plot shows five loss points, which are marked with ticks on the $x$-axis. The third loss is caused by packet corruption while the remaining losses are caused by congestion. In the absence of corruption losses, the *cwnd* naturally oscillates between $\frac{W}{2}$ and $W$, as expected. At point "A", however, the TCP experiences a corruption-based loss. This causes the *cwnd* to be reduced by half. When using $CETEN_P$, "A" is a decision point where the TCP flips a weighted coin. Regardless of the outcome of the coin flip, the TCP will continue transmission with no additional problem. However, that is not the case at point "B". When the TCP experiences loss at point "B", (the network's limit) and $CETEN_P$ flips the coin, there are two possible outcomes: congestion or corruption. When the coin lands on "congestion", the connection will reduce *cwnd* by half and continue transmission as shown in the diagram. When the coin lands on "corruption", the connection *will not* reduce *cwnd* and continue transmission. However, since *cwnd* is $W$, the saturation point of

[6] http://www.isi.edu/nsnam/

the network, and is not reduced, *additional losses will occur*. These losses will trigger additional coin flips until a coin flip eventually forces the TCP sender to reduce *cwnd*.

Figure 4 shows the limited utility of $CETEN_P$ in networks with little or no multiplexing. In other words, in some cases $CETEN_P$ ultimately has no choice on whether to reduce *cwnd*. Therefore, the fraction of the cases in which not reducing *cwnd* actually aids performance is less than predicted. Put another way, if 10% of the loss is due to corruption then the TCP sender will be able to avoid reducing *cwnd* by half on the order of 1% of the time. $CETEN_P$ may work better in networks with a higher degree of statistical multiplexing than that used in the simple simulations presented above. In such cases, when a congestion loss happens, it does not necessarily mean that any given TCP will be the predominant cause of the congestion. Hence, as long as some number of competing flows slow down, a particular $CETEN_P$ flow may not be forced to reduce *cwnd* on congestion-based loss.

## 4.2 Simulations with Competing Traffic

Our second set of simulations involves a more complex traffic pattern designed to explore CETEN in an environment with competing traffic. These simulations involve one TCP connection in each direction across our network. We provide results for only one of the two TCP connections in this paper, however we note that both TCP connections perform similarly. In addition, 5 on/off constant bit-rate (CBR) flows are constructed in each direction. These CBR flows have on and off times picked randomly from an exponential distribution with a mean on time of 2.5 seconds and a mean off time of 10 seconds. When on, each CBR flow transmits at 1 Mbps (one-fifth of the bottleneck bandwidth). The corruption rate is applied to all traffic in this set of simulations.

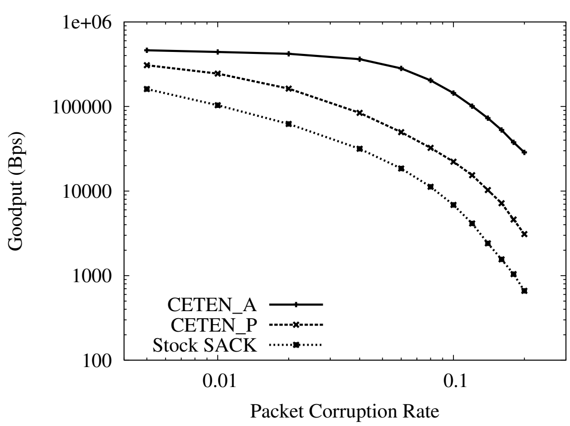



**Figure 5: TCP goodput in a congested network as function of the corruption rate.**

Figure 5 shows the TCP goodput obtained in simulations involving competing traffic as a function of the corruption rate applied to the network. The "Stock SACK" line shows the performance of stock TCP SACK. As in the last section, the performance drops as the total loss rate increases, as expected.

$CETEN_A$ shows largely the same characteristics in figure 5 as discussed in § 4.1, obtaining up to an order of magnitude better performance than stock TCP. The performance degradation of $CETEN_A$ as the corruption rate increases has three fundamental causes. First, a portion of the performance degradation is appropriate since the network is congested and therefore the TCP connection cannot expect to utilize the full capacity. The plot illustrates this point in showing that $CETEN_A$ does not utilize the full capacity even when there is little or no corruption-based loss (in contrast to the results shown in § 4.1). The second cause of the performance degradation is ACK loss. The $LEAST$ algorithm for estimating the total loss rate depends on receiving DSACK notifications from the receiver. When these ACKs are lost, the sender overestimates the total loss rate (which, in turn, decreases the MDF). The final cause of performance degradation for $CETEN_A$ is lost retransmits, as discussed in § 4.1.

As in the single flow tests, $CETEN_P$ shows improved performance when compared to stock TCP SACK while not enhancing goodput as much as $CETEN_A$. The suggestion in the last section that $CETEN_P$ would work better in an environment with some statistical multiplexing seems to be true. However, $CETEN_P$ is still susceptible to situations where the choice of whether or not to reduce *cwnd* is effectively taken away from the TCP sender.

## 4.3 Fairness Simulations

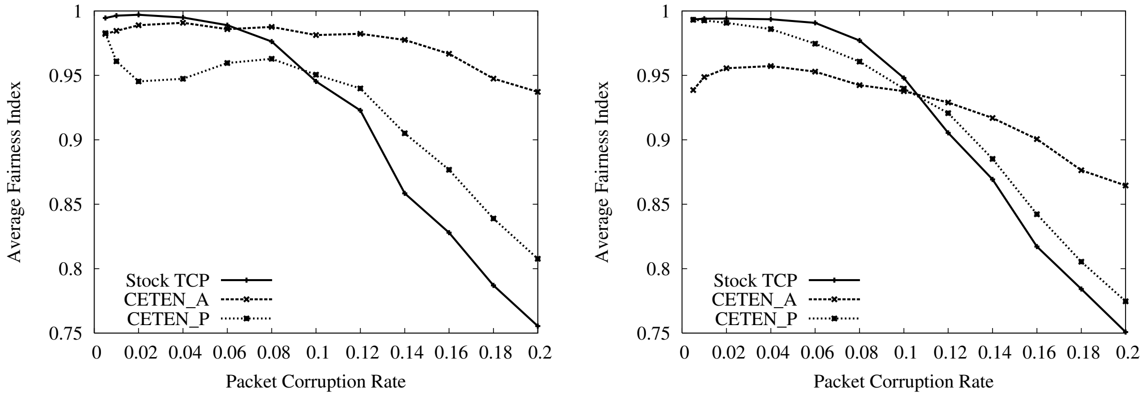
The simulations presented above show that both variants of CETEN offer performance benefits in networks with a non-negligible amount of corruption-based loss. We now turn our attention to a preliminary analysis of the dynamics of CETEN in the presence of more than one TCP connection sharing the network path. Over a common network path, TCP has been shown to converge to a state where all connections receive their "fair share", or about $\frac{1}{N}$-th of the available bandwidth when $N$ TCP connections are active (which our results also confirm). To investigate the impact of CETEN on this property of TCP, we use simulations that utilize the same setup used in the last two sections, with a different traffic mix. In this section we explore simulations with 10 and 50 competing TCP connections running in each direction across our simulated path. All connections run the same TCP variant. To assess fairness we use Jain's fairness index [17]. The index is computed as:

$$f(x_1, x_2, \cdots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \quad (3)$$

where $x_i$ is the total number of bytes transferred by connection $i$ and $n$ is the total number of connections in the simulation. A fairness index of 1.0 indicates that each flow transmits the exact same number of bytes.

Figure 6 shows the average fairness index for the 10 and 50 connection simulations as a function of the packet corruption rate. These plots lead to several observations:

- When the corruption rate is less than or equal to 10%, all TCP variants studied achieve fairness indices of roughly 0.95–1.0, indicating that the bandwidth is being shared roughly equitably amongst all the flows (in both sets of simulations).
- When the corruption rate is in the 0–10% range, stock TCP SACK is more fair than either CETEN variant at most points studied. The throughput difference between connections is due to CETEN's aggressiveness (and sometimes over-aggressiveness as will be shown in the next section) causing more spread in the performance attained. However, the reduction in fairness over stock TCP SACK does not represent a significant departure from the way stock TCP SACK shares the bottleneck.
- Corruption rates of greater than 10% cause degradation in fairness across all variants of TCP tested. As the packet cor-




(a) 10 Flows (b) 50 Flows

**Figure 6: Average fairness indices.**

ruption rate increases from 10% to 20%, stock TCP SACK's fairness index drops by roughly 0.2 (in both sets of simulations). As outlined in § 4.1, as the loss rate increases, TCP is more prone to relying on the retransmission timer (RTO) to repair loss. Once in this small *cwnd* regime, TCP's performance is more prone to vary based on the specific losses the connection experiences. Therefore, competing TCPs behave differently because they are exposed to various loss amounts and patterns – each of which has a large impact on performance. $CETEN_P$ generally tracks stock TCP SACK when the corruption rate is more than 10% because of $CETEN_P$'s susceptibility to being forced to reduce *cwnd* in high loss environments and therefore behave roughly as a stock TCP flow. $CETEN_P$ is not quite as prone to reducing *cwnd* as stock TCP SACK and therefore is able to keep the *cwnd* a little larger and show a slight increase in fairness. $CETEN_A$ is generally more aggressive than stock TCP SACK and so is not as likely to fall into the small *cwnd* regime where performance is dictated by specific loss events. Therefore, $CETEN_A$ is able to better keep its fairness high because all the connections benefit from staying out of the small *cwnd* regime..

Generally, the CETEN schemes have fairness comparable to (or only slightly less than) that of stock TCP at the lower error rates. At the higher error rates, $CETEN_P$ is slightly more fair than stock TCP, as both experience fast degradations in fairness, while $CETEN_A$'s fairness index is shown to be a bit more robust to high error rates.

## 4.4 Friendliness Simulations

The simulations presented in the last section show that both $CETEN_P$ and $CETEN_A$ leave the multiplexing abilities of TCP intact when TCP is competing only with like flows. In this section we explore a further question as to whether TCP senders using CETEN modifications compete in a "friendly" manner with stock TCP senders. We consider two different network conditions in this section. The first condition is when $e$ is high enough to ensure that the aggregate stock TCP SACK traffic from all flows on the network cannot consume the bottleneck bandwidth. The second case we explore is a case when the aggregate stock TCP SACK traffic can saturate the available capacity regardless of the corruption losses present in the network. We establish "ideal friendliness" as the average stock TCP SACK performance with no CETEN flows are present in the network. We would hope that the performance of stock TCP SACK would not deviate from this ideal when CETEN flows are introduced into the network.

### *4.4.1 Underutilized Network*

Our first experiment involves the same dumbbell topology used in our previous simulations. In the first set of simulations we initiated 50 TCP connections all from the same endpoint and applied a corruption rate of 1% to the data packets. The number of connections utilizing CETEN varies from 0–50 with stock TCP SACK senders making up the remainder of the connections. The simulations run for 5 minutes. The bottleneck bandwidth in this set of simulations is 100 Mbps (which is approximately twice the capacity that can be consumed by 50 stock TCP SACK flows with the given RTT and corruption rate).

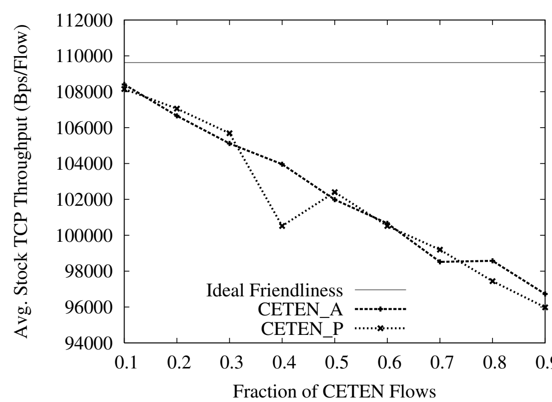



**Figure 7: Stock TCP throughput as a function of the fraction of CETEN flows in a network with a 100 Mbps bottleneck.**

Figure 7 shows the average throughput of the stock TCP SACK flows as a function of the fraction of CETEN flows used. The plot first shows that $CETEN_P$ and $CETEN_A$ perform similarly. Also, the plot shows that the average performance of the stock TCP SACK flows decreases as the number of CETEN flows increases (by roughly 10% when 90% of the flows use CETEN). The reason for the performance degradation is that CETEN is more effective in utilizing the available capacity. Therefore, CETEN generates congestion-based losses where there were none without CETEN. Therefore, the total loss rate is increased and $p$ is what determines stock TCP SACK performance. The point where 40% of the connections use $CETEN_A$ is clearly an outlier on the plot. We have not yet been able to nail down a cause of this anomaly. However, we note that the point is only a couple of percentage points different from where the point "should be" and therefore does not present a large problem for $CETEN_A$. We will continue to search for the source of this outlier.

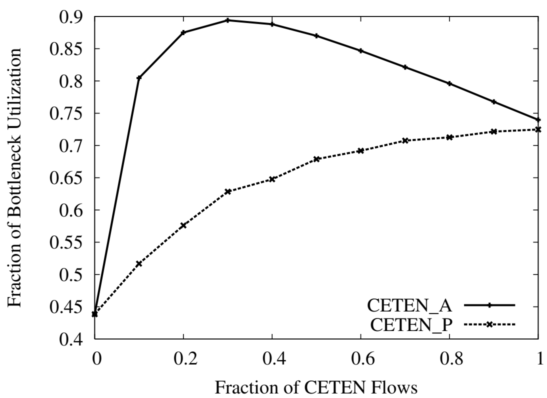



**Figure 8: Fraction of bottleneck utilized as a function of the fraction of CETEN flows.**

Figure 8 shows the bottleneck utilization as a function of the fraction of CETEN flows active in the network. This figure shows that the CETEN flows are able to better use the available capacity than the non-CETEN flows. In addition, figures 7 and 8 together show that CETEN is obtaining better performance largely by utilizing previously unused capacity rather than stealing capacity from non-CETEN connections.

Figure 8 also shows that $CETEN_A$ is able to increase utilization until approximately half the connections are using $CETEN_A$, after which the bottleneck utilization drops off. This suggests that $CETEN_A$ is overly aggressive because flows are increasing $p$ to the detriment of performance. $CETEN_P$ does not exhibit this problem because it is less aggressive than $CETEN_A$. The results do show that the over-aggressiveness of $CETEN_A$ does ($i$) improve overall performance and ($ii$) not greatly impact competing non-CETEN connections. Future work should include experimenting with ways to make $CETEN_A$ less aggressive and able to better use the bottleneck bandwidth (e.g., by using a different MDF function or a different set of shaping and bounding parameters).

### 4.4.2 *Fully Utilized Network*

The second experiment is similar to the first except that the bottleneck bandwidth is reduced such that the traffic pattern can fully utilize the bottleneck link. We ran two sets of simulations with the bottleneck bandwidth set to 5 Mbps and 25 Mbps. The packet corruption rate was 1% for both sets of simulations. Again, we vary the fraction of connections that use CETEN (with the balance using stock TCP SACK). This simulation scenario is a case when no mitigation for corruption-based losses is needed (or, appropriate) because ($i$) the network is fully utilized and ($ii$) the connections are all obtaining roughly the same performance (i.e., their "fair share"). Therefore, we'd like to determine whether CETEN causes the network to diverge from this "ideal" state by being overly aggressive.

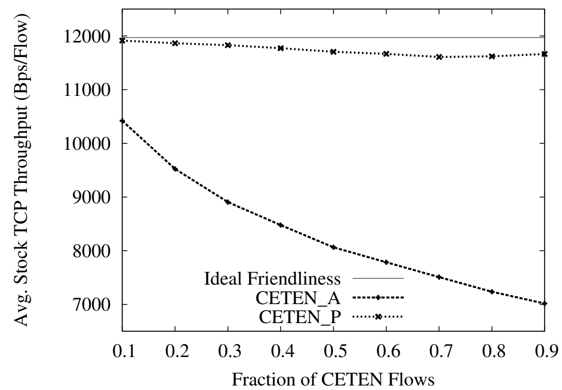



**Figure 9: Stock TCP throughput as a function of the fraction of CETEN flows in a network with a 5 Mbps bottleneck.**

Figure 9 shows the average performance of stock TCP SACK connections as a function of the fraction of competing CETEN connections in the case where no "spare" capacity exists for a 5 Mbps bottleneck. The utilization of the bottleneck link is nearly full (93–96%) in all simulations shown in the figure. In these simulations, each connection's "fair share" of the network is just under 4 segments per RTT (when taking into account the bandwidth, the RTT and 150 packet drop-tail queue in the router). Unlike the case of the overprovisioned network presented in the last section, $CETEN_P$ and $CETEN_A$ no longer perform similarly. The plot shows that $CETEN_A$ "steals" bandwidth from stock TCP SACK, causing up to an order of magnitude reduction in the average performance of a stock TCP SACK connection. This is caused by the aggressiveness of $CETEN_A$ creating additional loss for stock TCP SACK. Since each connection's fair share of the network is less than 4 segments, these additional losses cause loss to be repaired via TCP's RTO — which is costly from a performance standpoint. Meanwhile, $CETEN_P$ achieves roughly ideal friendliness. As illustrated in figure 9, $CETEN_P$ deviates from the ideal case by roughly 5% in the worst case (when 90% of the connections use CETEN). As we observed in previous simulations, $CETEN_P$ increases the congestion load on the network at times by failing to reduce the sending rate when congestion occurs. However, $CETEN_P$ is essentially forced into a congestion response when congestion is occurring and persistent. Therefore, in a highly contentious scenario as shown in figure 9, *cwnd* will be adjusted roughly as it is in stock TCP SACK. In cases where corruption-based loss occurs in a period when congestion is not occurring $CETEN_P$ is more aggressive than standard TCP and therefore steals a small amount of bandwidth from stock TCP SACK (as shown in the figure).

Figure 10 again shows the average performance of stock TCP SACK connection as a function of the fraction of competing CETEN connections, but with a 25 Mbps bottleneck in this case. As above, the bottleneck utilization is nearly fully utilized in these simulations. Also, in this set of simulations each connection's "fair share" is roughly 7 segments per RTT (again, taking into account the bandwidth, the RTT and the 150 packet drop-tail router queue). The results when using a 25 Mbps bottleneck are different than those found when using a 5 Mbps bottleneck above. In this case, stock TCP SACK is able to absorb the aggressiveness of $CETEN_A$ — due to the larger average congestion window. Losses in the regime illustrated in this figure do not automatically trigger the RTO, as in the previous experiments. $CETEN_P$ per-

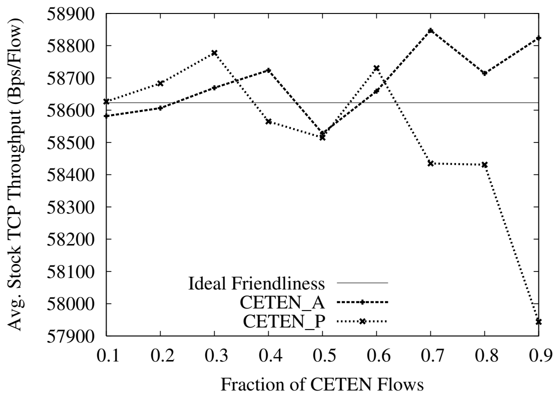



**Figure 10: Stock TCP throughput as a function of the fraction of CETEN flows in a network with a 25 Mbps bottleneck.**

forms similarly to the previous friendliness experiments and does not have a large impact on stock TCP SACK performance ($\pm$ 1% of ideal).

# 5. IMPLEMENTATION ISSUES

The previous section shows CETEN to be a promising method for mitigating the impact of corruption-based loss on TCP performance. However, there are several practical concerns to deploying CETEN in production networks. In this section, we present a first examination of some of these practical issues. This section is meant as an overview of items that will require careful consideration to implement CETEN. Future work will include a larger analysis of this space.

## 5.1 Information Transmission

As discussed in § 2 CETEN requires a method to gather $p$ and $e$. To use a system similar to the variant used throughout this paper, CETEN requires header space to encode the corruption survival probability. This header space most likely needs to be in the IP header and cannot be encrypted with IPsec [20] so that intermediate nodes can access and update the survival probability. A header option may be a viable approach (or an IPv6 extension header [8]). Alternatively, CETEN could re-use some of the current bits from the IP header (e.g., the IP ID field – also proposed by a number of researchers, such as [31]). Additionally, a TCP option to encode the survival probabilities and transmit them to the TCP sender in acknowledgments would also be required.

Additional methods to gather the needed information are outlined in § 2 and deserve consideration, as well — especially in the context of the practical that middleboxes introduce (outlined in § 2.3).

## 5.2 Router Assistance

CETEN requires assistance from routers and base stations in the network path between the sender and receiver. The additional requirements on routers are twofold: ($i$) additional state will potentially need to be kept by the router/basestation to calculate survival probabilities and ($ii$) additional processing will be required to calculate and insert survival probabilities into packets traversing the router/basestation.

The intermediate nodes in the network that intend to support CETEN flows will have to track packets dropped due to corruption – which is not a requirement of current routers. However, many current production routers do keep counters that track the number of corrupted packet arrivals and the number of total packet arrivals. Furthermore, keeping such counters is not a terribly onerous additional requirement to place on intermediate nodes that do not currently keep such information.

The second requirement on intermediate nodes is the processing time needed to update the survival probability in incoming packets. To soften this requirement somewhat, we note that (as discussed in § 5.4) global deployment of CETEN in all intermediate nodes is not necessary. Therefore, we believe that the majority of intermediate nodes will not implement CETEN because their connected links are not prone to corruption. While this removes concerns about CETEN's computational overhead in core routers, the nodes that do support CETEN will still have to accommodate all the CETEN-aware traffic traversing the node. Because each CETEN-aware node generates a corruption rate averaged over a relatively long period of time, the aggregate corruption rate of a particular, fixed network path will change only slowly over time. An individual TCP endpoint, therefore, need not query the network path with every data packet and would only need to send queries "every so often", greatly reducing CETEN-imposed demands on the nodes along the path. For instance, a requirement that a router deal with one CETEN request per connection every 10 seconds would not tax a router nearly as much as a CETEN scheme that requested a survival probability sample on every packet. Finally, we note that [13] shows that routers generally take very little time to generate ICMP messages, indicating that router's have the some ability to cope with requests (at least at a low rate). However, the cost may increase if the router has to process such requests for every TCP connection traversing its link(s).

## 5.3 Deriving the Corruption Rate

Another area of future work involves the exact method for arriving at the corruption rate at a particular intermediate node, including determining reasonable timescales over which to compute $e$. Is it enough to just calculate the corruption rate every "so often"? Should the intermediate node implement some sort of moving average? How long of a history should the node keep? Similarly, thought is required into how the TCP sender should use the $e$ estimate from the network. Should the TCP sender just use the given $e$? Should the TCP sender use some form of moving average on the $e$ samples received? Over what timescales should $p$ be estimated by the TCP sender? Should $p$ be a cumulative estimate since the beginning of the connection or some sort of average of a number of $p$ samples? These sorts of open questions are left as future work, but are likely important to answer to fully understand CETEN.

Additionally, non-point-to-point networks need further consideration. For instance, in some shared access systems, $e$ will vary between various end-hosts. Should the basestation just keep a single aggregate $e$ in this case? Or, should $e$ be tracked on a per-host basis? The answer to these questions is likely to be highly dependent on the underlying link layer technology — showing the necessity of further testing of CETEN in more realistic environments.

## 5.4 Incremental Deployment

CETEN as described in this paper (i.e., not $CETEN_O$) does not require all intermediate nodes to be CETEN-aware, which eases the task of incremental deployment. An intermediate node whose links experience only negligible corruption will simply multiply the survival probability in an arriving packet by roughly 1.0 – effectively leaving the cumulative path state unchanged. Also, when using CETEN, not all nodes that attach to corruption-prone links need to support CETEN for the scheme to be beneficial. The mitigation provided by CETEN is proportional to the number of intermediate hosts connected to corruption-prone links that support CETEN. Therefore, even a single CETEN-savvy intermediate node at a point of heavy corruption can be beneficial to TCP performance.

### 5.5 Security Implications

As discussed thus far, CETEN is vulnerable to an "attack" by the data receiver. If the receiver were to inform the sender that all the drops on the path were caused by corruption and not congestion, then the receiver could induce the sender into transmitting at an inappropriately high rate (to the point of effectively turning off congestion control). Such an attack would allow a receiver to obtain more than its share of the network resources at the expense of other connections sharing the path. This attack is similar in spirit to the schemes discussed in [30].

Unfortunately, this avenue of attack is fundamental to the design of CETEN since the receiver is the only entity in the path that can characterize the corruption status of the last link in the path. We believe that heuristics can be designed to detect egregious inflation of the corruption rate by the receiver. For instance, the sender could compare the reported corruption rate with the estimated total loss rate to attempt to detect cheating. Additionally, the sender could initialize the corruption survival probability to some random value rather than 1.0 in an effort to prevent the receiver from knowing where the value started; this would make it no longer straightforward for the receiver to estimate the loss rate and apply the correct transform on the probability to report that all losses were due to corruption. However, the system will likely retain a vulnerability to subtle gaming no matter what mechanisms are implemented given the fundamental requirement to trust the receiver to characterize the last hop in the path.

As alluded to in § 2, gathering $e$ reports from the intermediate nodes in the network opens these nodes up to an additional avenue for a denial-of-service attack. CETEN requires that the intermediate nodes do a small amount of work on behalf of the endpoints. However, an attacker could potentially leverage this "small amount" of work into a larger problem for the intermediate node by bombarding the node with requests for information.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper we sketch the CETEN mechanism in theoretical and practical terms and discuss preliminary experiments that show CETEN to be a promising idea. Both variants of CETEN show performance gains over stock TCP SACK without sacrificing fairness to like connections. While $CETEN_A$ cannot fully utilize a bottleneck in some situations, it does improve the utilization over stock TCP SACK. $CETEN_P$ does not improve performance as much as $CETEN_A$, but competes more fairly with non-CETEN traffic.

The work presented in this paper is preliminary. In a number of ways the simulations presented in this paper are idealistic in that they allow TCP to converge and the various network properties are somewhat constant. As an initial step this is a reasonable approach. However, additional simulations are needed to explore the parameter space, including: experiments with various MDF functions, with more complex and realistic traffic patterns and with more realistic corruption models. In addition, future work also needs to include tackling issues such as how often the transport should sample the corruption rate of the path, how the routers should calculate their corruption rate (and on what timescales) and how to mitigate the practical security concerns introduced by CETEN. Finally, the thorny issues discussed in § 5 require further study.

Finally, we note a more general class of future work on thinking about how much information the internal nodes should provide to the endpoints in a network. The end-to-end argument [29] suggests that the network be very simple and the "smarts" be located at the network edges. However, several recent proposals have suggested that internal nodes provide the endpoints with various pieces of information (e.g., ECN [28], XCP [18], QuickStart [16], CETEN). An overarching architectural question is how much of this information *should* be provided? And, if such information is provided, then what else might be useful for the network to provide (e.g., information about packet reordering or asymmetry)?

## Acknowledgments

Rajesh Krishnan, Craig Partridge and James Sterbenz developed the original CETEN algorithm (given in [21]) and provided useful feedback and assistance on the CETEN schemes developed for this paper. Mike Cauley provided key insight that led to the development of $CETEN_A$. We have given talks on CETEN at various places and received much useful feedback from a group of people too large to name. The anonymous CCR reviewers provided useful comments on this paper. The last author's work was funded by NSF grant number 0205519 and NASA's Glenn Research Center. Our thanks to all!

## 7. REFERENCES

[1] M. Allman, W. Eddy, and S. Ostermann. Estimating Loss Rates with TCP. *ACM Performance Evaluation Review*, 31(3), Dec. 2003.

[2] B. Arazi. *A Commonsense Approach to the Theory of Error Correcting Codes*. MIT Press, 1988.

[3] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[4] H. Balakrishnan and R. H. Katz. Explicit Loss Notification and Wireless Web Performance. In *Proceedings of IEEE Globecom Internet Mini-Conference*, Nov. 1998.

[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *ACM SIGCOMM*, Aug. 1996.

[6] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance Over Wireless Networks. In *ACM MobiCom*, Nov. 1995.

[7] J. Bennett, C. Partridge, and N. Shectman. Packet Reordering is Not Pathological Network Behavior. *IEEE/ACM Transactions on Networking*, Dec. 1999.

[8] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, Jan. 1996. RFC 1883.

[9] W. Eddy. Improving Transmission Control Protocol Performance with Path Error Rate Information. Master's thesis, Ohio University, Mar. 2004.

[10] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *ACM SIGCOMM*, Sept. 2000.

[12] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP, July 2000. RFC 2883.

[13] R. Govindan and V. Paxson. Estimating Router ICMP Generation Delays. In *Proceedings of Passive and Active Measurement*, Mar. 2002.

[14] M. Handley, J. Padhye, S. Floyd, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification, Jan. 2003. RFC 3448.

[15] V. Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[16] A. Jain and S. Floyd. Quick-Start for TCP and IP, Oct. 2002. Internet-Draft draft-amit-quick-start-02.txt.

[17] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. Wiley, 1991.

[18] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *ACM SIGCOMM*, Aug. 2002.

[19] C. Kent and J. Mogul. Fragmentation Considered Harmful. In *ACM SIGCOMM*, Oct. 1987.

[20] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, Nov. 1998. RFC 2401.

[21] R. Krishnan, M. Allman, C. Partridge, and J. P. Sterbenz. Explicit Transport Error Notification (ETEN) for Error-Prone Wireless and Satellite Networks. Technical Report TR-8333, BBN Technologies, Mar. 2002.

[22] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options, Oct. 1996. RFC 2018.

[23] A. Medina, M. Allman, and S. Floyd. Measuring Interactions Between Transport Protocols and Middleboxes. In *ACM SIGCOMM/USENIX Internet Measurement Conference*, Oct. 2004.

[24] J. C. Mogul and S. Deering. Path MTU Discovery, Nov. 1990. RFC 1191.

[25] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, Sept. 1998.

[26] J. Postel. Internet Control Message Protocol, Sept. 1981. RFC 792.

[27] J. Postel. Transmission Control Protocol, Sept. 1981. RFC 793.

[28] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP, Sept. 2001. RFC 3168.

[29] J. Saltzer, D. Reed, and D. Clark. End-to-End Arguments in System Design. In *Proceedings of the Second International Conference on Distributed Computing Systems*, pages 509–512, Aug. 1981.

[30] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *Computer Communication Review*, 29(5):71–78, Oct. 1999.

[31] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, Sept. 2000.

[32] W. Stallings. *Data and Computer Communications*. MacMillan, 4 edition, 1994.

[33] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol, Oct. 2000. RFC 2960.

# APPENDIX

# A. ALTERNATE CORRUPTION MODELS

The body of this paper utilizes a uniform random process to create corruption-based losses for studying CETEN. While not realistic, we believe that the key *trends* and *insights* gained from these simulations are likely to hold across alternate corruption scenarios. As evidence, we used *ns*' two-state Markov corruption model to examine CETEN in the face of bursty corruption loss. In this model, all segments arriving during the "on" period are corrupted, while all segments arriving in the "off" period are not corrupted. The length of each "on" and "off" period is measured in time and determined using an exponential process with means of $X$ and $Y$ (defined below) respectively.

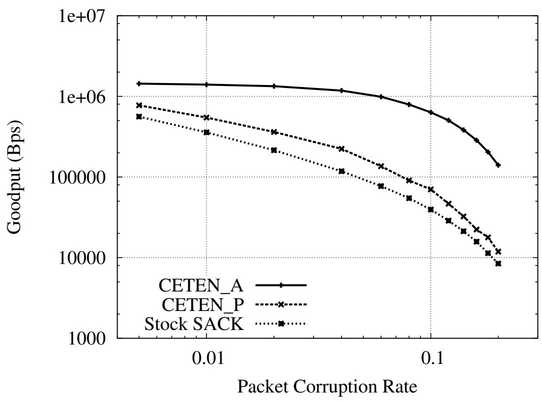
We repeated the simulations presented in § 4.1 with four sets of simulations with the average "on" period, $X$, defined based on the bottleneck serialization time corresponding to 1, 4, 8 and 16 segments (or, 2.4 ms, 9.6 ms, 19.2 ms and 38.4 ms). The average "off" period, $Y$, for a given simulation is then calculated to provide the desired corruption rate, $R$, as follows:

$$Y = (X + f)\left(\frac{1-R}{R}\right) \tag{4}$$

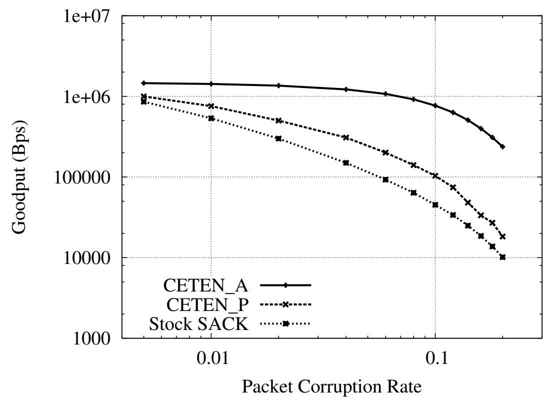
With $f = 0$ in the above equation, the link spends the proper fraction of *time* in the "on" and "off" states for the given $R$. However, the actual measured corruption-based packet loss rate is higher than $R$ (for all $X$ values we use). The key observation is that each "on" period does not directly correspond to a given number of segments. For instance, consider an "on" period to be randomly chosen as 2.4 ms — exactly the serialization time of a single segment. Chances are low that a single segment will arrive just as the "on" period starts. Chances are much greater that when the "on" period starts, the latter portion of some segment will be corrupted, followed by the corruption of the earlier parts of the following segment. That is, when the average "on" period is $N$ segments then chances are that $N + 1$ segments will be corrupted. Therefore, in determining the average length of the "off" period we need to take this into account by defining $f$ as the serialization time of a single segment (or, 2.4 ms in our simulations).

Figure 11 shows the performance of single TCP flow through networks with different corruption burstiness properties. As in figure 3 from § 4.1, all points shown are the result of 30 random simulations. Further, the corruption rate observed is verified to be within 10% of the desired corruption rate. The results in the figure show the same basic trends that are present in the case when using the uniformly distributed corruption model. $CETEN_A$ obtains the best performance, followed by $CETEN_P$, with stock SACK TCP showing the worst performance. In addition, we also note that each TCP variant's performance drops off as the corruption rate increases. Finally, the shape of the dropoff is different depending on the corruption pattern employed — indicating that the burstiness of the loss does have an impact on TCP dynamics.

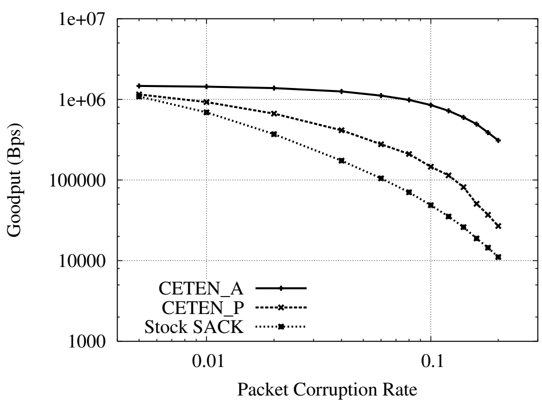
While there are differences between the Markov-based corruption model and the uniform-based corruption model used in the rest of this paper, we believe the results based on the uniform corruption model are reasonable to gain an initial understanding of the general trends CETEN provides. That said, the results in this appendix clearly show that work on the particular implications of CETEN in networks with specific link layer technologies is a rich area for future work.



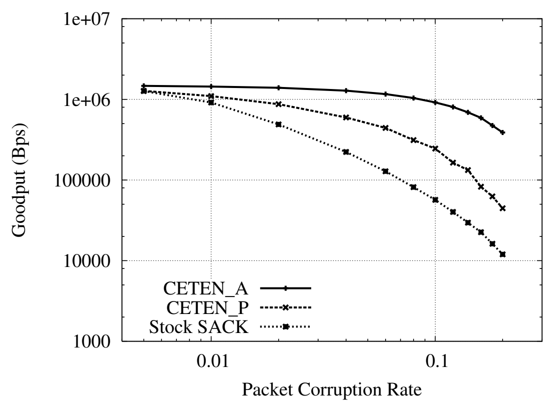

(a) Avg. Corruption Time = 2.4 ms (≈ 1 segment).




(b) Avg. Corruption Time = 9.6 ms (≈ 4 segments).




(c) Avg. Corruption Time = 19.2 ms (≈ 8 segments).




(d) Avg. Corruption Time = 38.4 ms (≈ 16 segments).

**Figure 11: Throughput of a single TCP flow as a function of the corruption rate for various corruption models based on two-state Markov models.**