SMART Retransmission: Performance with Overload and Random Losses

S. Keshav S. P. Morgan
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974
{keshav,spm}@research.att.com

Abstract

Feedback flow control, in conjunction with limited buffering in the network, inevitably leads to packet loss. Effective congestion control requires not only intelligent flow control but also a good retransmission strategy. We present a new retransmission strategy called SMART (Simple Method to Aid ReTransmissions), that combines the best features of the traditional Go-back-N (GBN) and selective-ack strategies.

We show, first, that GBN retransmission with static window flow control leads to congestion collapse when the nominal load exceeds the link capacity. Second, we can avert congestion collapse by replacing GBN with SMART retransmission, even with static window flow control. Third, SMART retransmission, when combined with intelligent flow control, performs extremely well, both when losses are due to buffer overflows and when losses are random. Moreover, sources implementing intelligent flow control as well as SMART have a substantial advantage over static-window SMART sources under overload. Finally, in the presence of heavy random losses, SMART plus intelligent flow control is far better than TCP.

1. Introduction

Most current approaches to congestion control focus on the problem of flow control, that is, techniques that enable an end-system to match the currently available service rate in the network.While flow control is an important problem, it often overshadows the equally important choice of retransmission strategy. If losses due to line errors (as in wireless networks) or due to buffer overflows (as with network congestion) are common, a poor retransmission strategy can lead to congestion collapse. It is important, therefore, to develop robust and efficient retransmission strategies. In this paper, we present SMART (Simple Method to Aid ReTransmissions), and use exhaustive simulations to show that it satisfies both the robustness and efficiency criteria. While SMART was developed in the context of a rate-based flow control protocol, it is equally applicable to window-based protocols. We will show that SMART retransmission, in conjunction with intelligent flow control, not only performs extremely well with congestive losses, but also performs nearly ideally in the presence of random losses. In contrast TCP's performance grows rapidly worse in the presence of frequent random losses.

The rest of the paper is organized as follows. Section 2 describes the SMART retransmission policy. Section 3 studies the baseline case with GBN retransmission and static window flow control. Section 4 shows that adding SMART to the baseline case avoids congestion collapse. In Section 5 we describe how a combination of SMART and intelligent flow control allows a source to not only avoid congestion collapse, but also obtain nearly optimal performance. Section 6 is a brief description of TCP's retransmission policy and an analysis that shows that TCP will behave poorly when the loss rate is high or the bandwidth-delay product is large. Section 7 shows that SMART + intelligent flow control behaves nearly optimally in the presence of heavy random losses. Finally, Section 8 presents our conclusions.

3. SMART Strategy

SMART is a novel retransmission strategy that combines aspects of the two traditional techniques: Go-back-N and selective-retransmit. We first outline the traditional schemes and their drawbacks, in order to motivate the design for SMART.

Go-back-N

In GBN, a source keeps track of an *error control window*, which is the set of sequence numbers that it has transmitted, but the receiver has not acknowledged. Typically, the receiver writes the cumulative acknowledgment (ack) number in the ack header: this is the last in-sequence packet it has received, and is the one smaller than the left edge of the error control window. In strict GBN, on a timeout, the source retransmits the entire outstanding window [SCHW77]. This has the advantage of being conservative, because the source treats every loss as a burst loss, but uses bandwidth inefficiently. Moreover, loss recovery depends on a timer, which is usually hard to set [ZHAN86]. If the timer is set too low, the source retransmits packets unnecessarily; if set too high, it wastes time, waiting for a timeout.

Some of this inefficiency is avoided by modified versions of GBN, as in TCP [WRIG95]. In these schemes, on a timeout, the source marks the entire window eligible for retransmission, but flow control ensures that the source retransmits only the first packet in the window. If incoming acks move the left end of the error control window, the source unmarks packets formerly marked eligible for retransmission. Thus, it is unlikely that the source would retransmit the entire error-control window. For example, suppose the error control window is 6-10, and the actual packet lost is 6. On a timeout, the source retransmits the packet with sequence number 6, and flow control ensures that the source sends no other packet for the next round trip time. If the ack moves the left edge of the error control window to 10, then the source does not retransmit packets 7-10. This scheme is clearly more efficient in its use of bandwidth. However, this comes at the expense of reduced throughput during the first round-trip-time after a loss. Besides, the source still depends on a timer to initiate retransmission.

Selective acknowledgement

Several selective-acknowledgment schemes have been proposed in the literature [DOSH93]. In these schemes, every acknowledgment carries a bit-mask of the packets seen so far. Since the sender knows which packets have been received, only packets that are actually lost are retransmitted. Thus, selective-acknowledgment allows a source to use bandwidth efficiently. Besides, there is no need for a timer to decide that a retransmission is necessary. However, every ack header must carry the bit-mask, which can be a significant overhead if the window size is large. Moreover, the scheme incurs this bandwidth overhead even if there is no packet loss, which is the common case.

SMART

The SMART retransmission strategy seeks to be as efficient as selective-ack in its choice of packets to retransmit, and as conservative as GBN in the presence of burst losses. We would like to avoid using timers to the extent possible. Finally, we would like to achieve our objectives without coupling flow and error control.

The key idea in SMART is to build the bit-mask of correctly received packets at the sender, instead of carrying it in the ack header. Each ack therefore carries two pieces of information: the cumulative ack, as in standard GBN, and the sequence number of the packet that caused the ack to be initiated. As we will show below, the second piece of information not only allows the sender to know which packets have been correctly received, but also allows the sender to correctly infer which packets have been lost, and to take corrective action, that, for the most part, does not require the intervention of a timer. This relative independence from timers is the key to its success — as Zhang has shown, retransmission protocols that are strongly dependent on timers have severe problems [ZHAN86]. However, we do use timers as a mechanism of last resort. Thus, in the common case, SMART is as efficient as selective ack, and in the worst case, it is as conservative as GBN. We remark here that the scheme is particularly well suited to wireless environments that are characterized by sudden burst losses.

Detailed description

We now present a detailed description of SMART. For the moment, assume that every packet is acked. When a source receives an ack, it compares the cumulative acknowledgment with the sequence number of the packet that caused the acknowledgment. If the two are not the same, then the source guesses that all the packets in the sequence space between the cumulative ack and the packet being acked are lost, and places them in the transmission queue for subsequent retransmission. For example: if the source receives an ack that has a header (2,7), where 2 is the cumulative ack, and the ack was caused by the receipt of a packet with sequence number 7, then the source guesses that packets with sequence numbers 3, 4, 5, and 6 are lost. It may be that these packets are actually misordered. SMART is aggressive about retransmission, and so when it is uncertain whether a packet is misordered or lost, it always guesses that the packet is lost, and retransmits it. (Note that this is similar to the *fast retransmission* policy in TCP Reno, except that in TCP Reno, three duplicate acks have to be received before the source initiates a fast retransmission.)

There is one exception to this rule. If a source has already done a fast retransmission of a packet, then it will not be retransmitted by the receipt of a subsequent ack. For example, if acks (2,7), and (2,9) arrive to a source, on receiving the first ack, the source retransmits 3, 4, 5, and 6. However, on receiving the second ack, the source only retransmits 8, because packets 3-6 have been retransmitted earlier. This rule catches almost all packet losses. It fails only when the fast retransmission itself is lost. In order to detect this, we use a different heuristic, described next.

Lost fast retransmissions

The intuition behind the scheme is that if a fast retransmission succeeds, then the cumulative ack should increase one round trip time after the fast retransmission was sent. Otherwise, the fast retransmission must have been lost. Suppose that the source sends a packet with sequence number P at time $t(P)$ that is lost in the network. If the source sends packet $P+1$ at time $t(P+1) = t(P)+\varepsilon$, and the round-trip-time is T_{RT}, then at time $t(P)+\varepsilon+T_{RT}$, the source can discover that P is lost (because the cumulative ack and sequence number for this ack differ). Thus, the source retransmits P at time $t+\varepsilon+T_{RT}$. In the interval $[t+\varepsilon+T_{RT},\ t+\varepsilon+2T_{RT}]$, the cumulative acks received by the source cannot increase, since the lost packet causes a hole in the sequence space at the receiver, that is filled only when the retransmission is received. However, after time $t+\varepsilon+2T_{RT}$, if the fast-retransmission indeed reached the receiver, the cumulative acknowledgment received by the source should have increased. In fact, if the cumulative ack received after time $t+\varepsilon+2T_{RT}$ is not larger than the cumulative ack received at time $t+\varepsilon+T_{RT}$, then the fast-retransmission must surely have been lost. Of course, we need to allow for some variation in the round-trip-times, to allow for variable queueing delays. However, this logic lies at the basis of two algorithms to correct for most lost fast-retransmissions that we discuss next.

In the first algorithm, every time the source sends a fast retransmission, it sets a timer to expire one round-trip-time from the current time. If the cumulative acknowledgment does not increase by the time the timer expires, the packet with sequence number one larger than the cumulative-ack is retransmitted. In order to account for variations in the round-trip-time, the timer should be set to a value somewhat larger than the measured smoothed round-trip-time. This scheme allows us to reliably detect lost retransmissions. However, it not only requires a fair amount of work in setting and resetting timers, but also is dependent on accurate measurement of round-trip-times for its success.

We can avoid these problems at the expense of another field in the packet header.[1] In the second algorithm, the source maintains an *id* state variable that is guaranteed to be monotonically increasing with time (unlike the sequence number field, which can decrease because of retransmissions). We also add an id field to the transport-layer packet header. Every time the source sends a packet, it writes the current id value in the packet header, and increments the state variable. The id field is copied into the acknowledgment by the receiver. The id field allows the source to measure round trip times without timers. For example, if the source sends a packet with id 5 at time *t*, then it should receive an ack with id 5 one round trip time later.

Recall that the source would like to measure one round-trip-time from the time it sends the fast retransmission. Thus, when it sends a fast-retransmission, it stores the current cumulative-ack and the id of the fast-retransmission packet in two state variables. On receiving an ack, if the id of the packet is larger than the stored id, but the cumulative ack is the same as the stored value, then the fast retransmission must have been lost. For example, if the source does a fast retransmission of packet 52 (cumulative ack is 51) when the next packet would have id 213, if the cumulative ack is still 51 when an ack with id 213 is received, the source retransmits 52.

If a source loses a fast retransmission, the line is probably quite lossy, because the source has lost the

[1] This version was used in the simulations of the present paper.

same packet twice. To avoid losing the packet three times, the source thus places *two* copies of the same packet in the retransmission queue. This ensures that the sender will make progress (since the cumulative ack is very likely to increase — it won't increase only if *both* of the retransmissions are lost) even on very lossy lines.

Thus far, we have discussed how to recover from lost packets, and lost retransmissions, without using timers. However, timers are always needed, if only to deal with the worst case, where all packets and all acks are lost. Thus, SMART uses timers as a recovery mechanism of last resort. The source measures round trip times and exponentially averages them as in TCP. On packet transmission, a per-connection timer is set to twice the smoothed round-trip-time estimate. On a timeout, the entire window, other than packets that are known to have been correctly received, is retransmitted. The timeout thus correctly retransmits lost packets even when the heuristics described above fail.

Error-control window size

We distinguish between the *error-control* window and the *flow-control* window at a source. The error-control window W_{EC} is the largest number of out-of-sequence packets that can be buffered at the receiver. For a window-controlled source, the flow-control window W_{FC} is the largest number of packets that the source may send without an acknowledgment. Limiting the flow-control window allows us to implicitly control the transmission rate of a connection, since the connection's rate is restricted to one flow-control window worth of packets per round-trip-time.

There are two requirements on the flow-control window: 1) It must be at least as large as the round-trip window W_{RT}, defined as the bandwidth-delay product of the connection. Otherwise the source cannot utilize the full speed of the circuit when there is no congestion. 2) It must not be larger than the error-control window, otherwise the receiver can't hold all the out-of-sequence arrivals that it might need to rearrange. If the round-trip window is less than the error-control window, as is almost always the case in practice, then the flow-control window can have any value in between.

For window-controlled circuits, where the source may transmit at full speed until the flow-control window is exhausted, too large a flow-control window runs the risk of overflowing the buffer space at bottleneck nodes. On the other hand, for rate-controlled circuits a sufficiently large error-control window permits the source to continue sending at the rate determined by the rate-control algorithm while the system recovers from packet losses.

Call the time between transmitting a packet and receiving its eventual acknowledgement, perhaps after multiple losses and retransmission, the *recovery time*. If the bottleneck has a rate μ, and the recovery time is T, then in order to keep the bottleneck busy, the source should be able to send $T\mu$ packets without waiting for a packet to be acked. In other words, the error-control window should be at least as large as $T\mu$.

For the SMART retransmission strategy, we argued in the previous subsection that if the source transmits a packet at time 0 and the transmission fails, the source is aware of the failure at time $T_{RT} + \varepsilon$. If the first fast-transmission fails, the source is aware of the failure at time $2T_{RT} + \varepsilon$. A simple inductive argument shows that as long as losses are isolated, that is, no loss occurs while the source is recovering from a loss, if the same packet is lost x times, then the recovery time is just greater than $(x+1)T_{RT}$, and the error-control window needs to be at least $(x+1)W_{RT}$. In principle, we cannot bound the number of times a retransmitted packet is lost. However, increasing the error-control window beyond a certain size provides rapidly diminishing returns, since the events that require very large error-control window sizes are very unlikely.

If losses can occur while the source is recovering from an earlier loss, then the analysis is substantially more complicated, since the recovery time depends in detail on which packets are lost, and on the order in which the losses happen. For example, if two packets are lost back to back (a burst of length 2), then with our scheme, the recovery time is nearly identical to the recovery time with a single loss. On the other hand, if the second loss happens halfway through the recovery from the first loss, the recovery time is substantially longer than the recovery time with a single loss. A detailed analysis of the recovery times for various combinations of losses and their relative probability of occurrence would allow us to

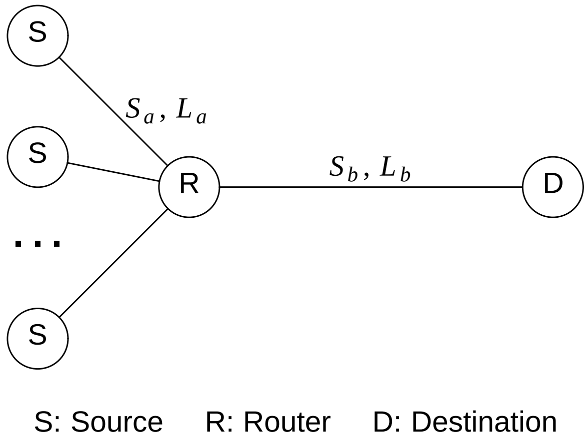


Figure 1. Simulation scenario.

select a ‘‘good’’ error-control window size, that is, a window size that balances the benefit of dealing with the most likely loss events with the cost of additional buffering at the receiver. However, this analysis would be rather complicated. Instead, we have studied the effect of window size on throughput through simulations (Section 7). We recommend that for normal operation, the error-control window be set to as large a value as possible, at least $2W_{RT}$, and preferably close to $4W_{RT}$.

2. Baseline case

In order to measure the effectiveness of SMART, we first study the performance of a *baseline* scenario shown in Fig. 1. In this scenario, packets from sources on access lines of speed S_a and latency L_a are multiplexed at a Fair Queueing router, which puts them onto a backbone link of speed S_b and latency L_b. The router has a fixed total buffer size and maintains a separate logical queue for each source. When a packet arrives that would cause the total buffer space to overflow, the arriving packet is discarded. The destination acknowledges every received packet with an acknowledgment of length *A*. Sources measure round-trip-times from the acknowledgments, smoothing them with an exponential averaging filter, as in TCP [WRIG95]. The timeout value is set to twice the smoothed round-trip-time. The sources do GBN retransmission of the entire outstanding window on a timeout, simultaneously doubling the timeout value.

After a source has generated a message containing some number of packets, it waits until all the packets have been correctly acknowledged. It then goes OFF for an additional, randomly chosen ‘‘think time’’, after which it again goes ON and repeats the cycle. In the simulation model, ON times are geometrically distributed with mean T_{ON} (each ON period generates an integral number of packets). OFF times are exponentially distributed with mean T_{OFF}. For simplicity we assume that the link between the router and the destination is the bottleneck, that is, $S_b \le S_a$, and we assume that during an ON period, a source generates bits at a rate equal to the rate of the bottleneck link, that is, $R_{ON} = S_b$. For the average length of an ON period, we take $T_{ON} = 2T_{RT}$, where T_{RT} is the round-trip time when only one source is active, i.e., there is no contention. Thus the average message length is $2S_b T_{RT} = 2W_{RT}$.

The nominal offered load r_1 due to a single source is equal to the average utilization of the bottleneck link when only one source is present. Thus,

$$r_1 = \frac{T_{ON}}{T_{ON} + T_{RT} + T_{OFF}} . \qquad (1)$$

We can vary r_1, within limits, by varying T_{ON} and T_{OFF}. The nominal offered load due to *N* statistically identical sources is Nr_1, and this can made as large as desired by suitable choice of *N*. The carried load cannot exceed 100% of the capacity of the bottleneck link, and the goodput is less than 100% if any packets have to be retransmitted.

For the simulations, we chose:

$$P = 500 \text{ bytes} = 4000 \text{ bits} ,$$

$$A = 40 \text{ bytes} = 320 \text{ bits} ,$$

$$T_{RT} = 0.05 \text{ s} ,$$

and for the number of packets in a round-trip window,

$$S_b T_{RT} / P = 100 \text{ packets} .$$

The above numbers give a reasonably large number of packets in a round-trip window and still achieve reasonable running times for the simulations. Translation from simulator numbers to real-world numbers is straightforward. An ATM cell contains 53 bytes, including payload plus header. Assuming a transcontinental round-trip time of 50 ms, at a backbone speed of 1.5 Mb/s the round-trip window size is 177 ATM cells . At 622 Mb/s, the round-trip window is 73,350 ATM cells. The size of a simulated packet is 1/100 of the round-trip window size.

We have considered three values of router buffer size, namely one round-trip window, 0.5 round-trip window, and 0.25 round-trip window. Each of the plots includes three curves, one for each value of the ratio *B* of buffer size to W_{RT}.

In the simulations of congestion loss, we set the error-control window equal to 600 packets, that is, 6 times the round-trip window. Such an window is essentially infinite for this scenario. The effects of error-control window size in a random-loss scenario are considered in Section 7.

We ran all the overload simulations with 10 statistically identical sources, that is, $N = 10$. Each simulation run lasted for 400 simulated seconds after an initial ''transient'' period of 20 seconds. Since one round-trip time is 0.05 seconds, each 20-second interval corresponds to 400 round-trip times. During each interval, the bottleneck link could have carried about 40,000 packets at full utilization. We varied the nominal offered load of a single source from 0.05 to 0.5, so that the total nominal offered load on the bottleneck link varied from 0.50 to 5.00. The ''fair share'' of a single source varied from 0.05 to 0.10, inasmuch as under congestion no source would be expected to get more than 1/10 of the available bandwidth on average. Thus we would expect a typical source to have a goodput of between 10 and 20 average-length messages per 20-second interval We considered this adequate for statistical purposes.

To estimate the reliability of the simulator output, we went through the usual procedure of taking a sample of n values of goodput corresponding to $n = 20$ intervals. We computed a sample mean $\bar{X}$ and variance $\hat{\sigma}^2$, according to [LAW82],

$$\bar{X} = (1/n) \sum_{i=1}^{n} X_i \ . \qquad (2)$$

$$\hat{\sigma}^2 = \frac{1}{n(n-1)} \sum_{i=1}^{n} (X_i - \bar{X})^2 \ . \qquad (3)$$

The intervals $(\bar{X} - 1.96\hat{\sigma},\ \bar{X} + 1.96\hat{\sigma})$ are plotted on Fig. 2. If the intervals are uncorrelated and the sample is large enough so that its mean is normally distributed, then the plotted intervals are ''95% confidence intervals''.

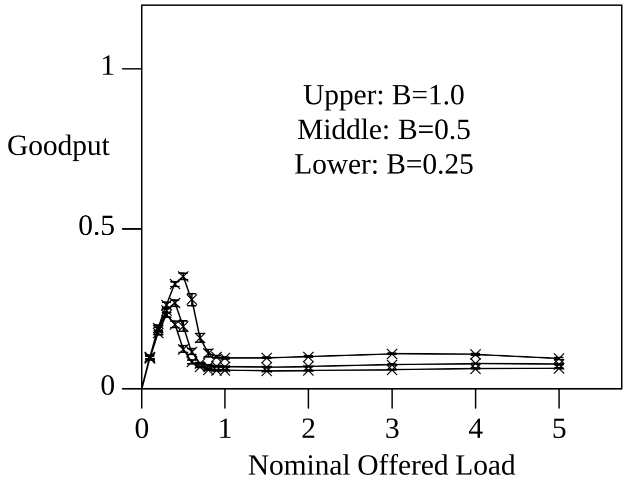


Figure 2. Goodput vs. offered load for 10 GBN sources.

Figure 2 shows the goodput as a function of the nominal offered load. The maximum goodput is between 25 and 35%, at an offered load of about 50%. As the load approaches 1, the goodput decreases dramatically, displaying classic congestion collapse. This is because GBN retransmits an entire window on a single packet loss. As the load increases, the probability of a buffer loss increases, decreasing the goodput. Note that, as might be expected, both the peak goodput and the load at which this peak is achieved, decrease with a decrease in the buffer size.

4. Baseline case with SMART retransmission

We saw that static window flow control, in conjunction with GBN retransmission, leads to congestion collapse. One way to deal with this problem, as in TCP, is to shut down the flow control window on a loss, thus reducing the load. However, *without* changing the flow control algorithm, if we simply replace GBN with SMART, we find that the congestion collapse disappears. Figure 3 shows the goodput as a function of the nominal load for a simulation scenario identical to the one above. It is clear that SMART prevents congestion collapse as the nominal load increases beyond 1.0. The achievable goodput is smaller with decreasing buffer sizes, but even with a nominal load of 500%, there is no decrease in the achievable throughput. This is a clear indication of SMART's effectiveness in dealing with buffer losses.

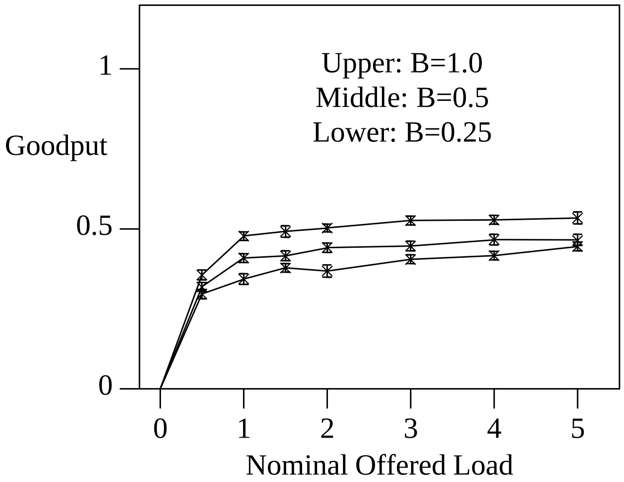


Figure 3. Goodput vs. offered load for 10 static-window + SMART sources.

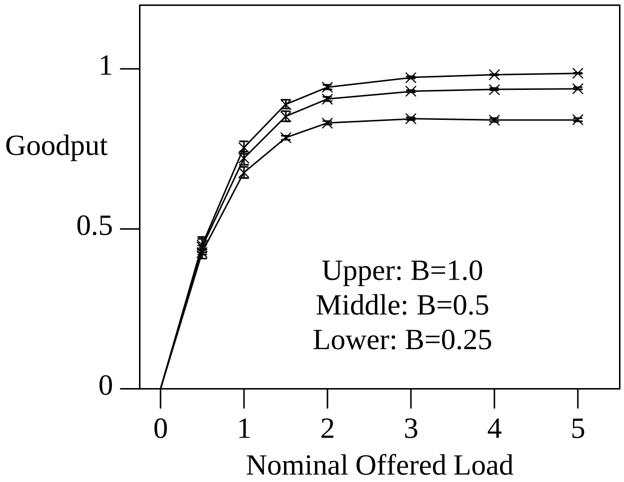


Figure 4. Goodput vs. offered load for 10 Packet-Pair + SMART sources.

5. Baseline case with SMART retransmission and Packet-Pair flow control

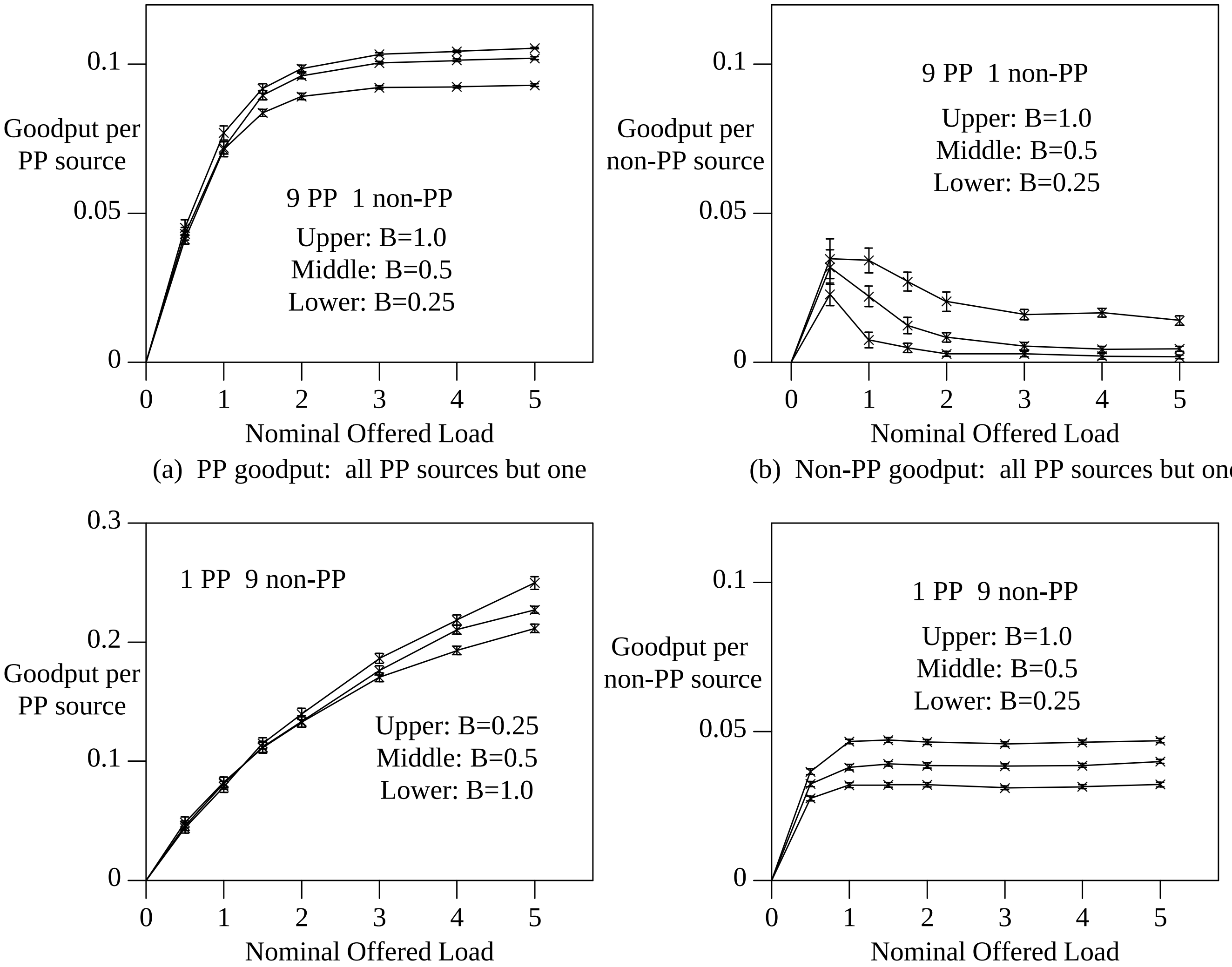
SMART prevents congestion collapse, but does not prevent packets from being lost in the first place. This is why, in Fig. 3, the achievable goodput is around 50%, instead of around 100%. In order to achieve better goodput, we not only need smarter retransmission, but also better flow control. In this section, we study sources that implement both SMART, and Packet-Pair flow control [KESH91, KESH95]. Packet-Pair is a rate-based flow control scheme designed for networks of round-robin servers. It measures the bottleneck service rate by sending all data in the form of back-to-back pairs, and measuring their inter-ack spacing. A filtered version of the measured rate is used in conjunction with a control law to ensure that the bottleneck's buffers are neither too full, nor too empty. If the network does not provide round-robin queueing but does give explicit rate feedback through the use of RM cells [BONO95], a Packet-Pair source would simply use this rate instead of making its own rate estimates. A detailed study of Packet-Pair performance can be found in [KESH95].

Figure 4 shows the goodput achievable with Packet-Pair + SMART, in the same scenario as before. Comparing this figure with Figs. 2 and 3, it is clear that sources implementing both PP and SMART not only avoid congestion collapse, but also get close to 100% goodput even when the buffer size is only a fourth of the bandwidth-delay product. For a fixed offered load, the goodput decreases, but not very fast, as the buffer size B decreases. This result helps to quantify the widespread belief that dynamic rate control should work satisfactorily with a total of less than one round-trip window of buffering. With PP, the buffer size needs to be at least $2N$ packets for N connections. In the present simulations, $2N$ packets would correspond to $B = 0.20$. Using smaller packets, SMART + Packet-Pair would undoubtedly function with still lower values of B. In an ATM network using AAL5, the spacing between cells of a single AAL frame could be measured at the receiver, reducing the buffer requirements still further.

We conclude this section by studying the behavior of mixtures of PP and non-PP sources when both are using SMART. The non-PP sources are ON-OFF with static window flow control, that is, they are identical to the sources in Section 4. The simulation scenario is the same as in Section 3.

The results of the simulations are plotted in Fig. 5. We have considered two combinations, namely 9 PP and 1 non-PP source, and 1 PP and 9 non-PP sources. Each plot shows per-source goodput, for each type of source, vs. nominal offered load,

Figure 5 shows that in general, when PP and non-PP sources are mixed, the PP sources capture more bandwidth per source than the non-PP sources, both when the PP sources are in the majority (Figs. 5(a)(b)) and when they are in the minority (Figs. 5(c)(d)). This is because they gain goodput at the expense of the non-PP source (to see this, compare



(a) PP goodput: all PP sources but one

(b) Non-PP goodput: all PP sources but one

(c) PP goodput: all non-PP sources but one

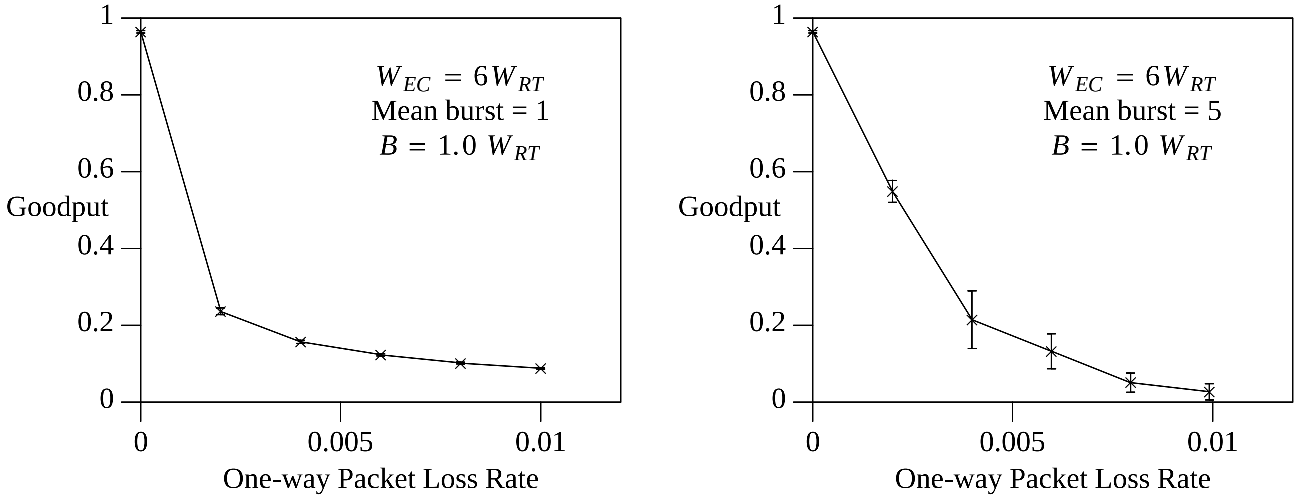
(d) Non-PP goodput: all non-PP sources but one

Figure 5. Goodput vs. offered load for PP and non-PP sources + SMART.

panel 5(a) with Fig 3). Under heavy load, several PP sources can essentially drive out a non-PP source.

When a single Packet-Pair source shares a link with many non-Packet-Pair sources, Figs. 5(c) and 5(d) show that the Packet-Pair source gets substantially more than its share, which would have been 10% of the congested link under the present scenario. Furthermore, the single Packet-Pair source actually does better with a smaller total buffer (Fig. 5(c)), because it knows how to behave while the non-Packet-Pair sources do not. These results clearly indicate that while SMART can prevent congestion collapse, by itself, it is not sufficient to *efficiently* use the link. To do so, it must be used in conjunction with an intelligent flow control algorithm that explicitly tries to match its sending rate to the bottleneck capacity. If used with a naive flow control scheme, SMART can reduce the number of retransmissions, but packet losses will still occur because of rate mismatches. The implication is clear: with Fair Queueing routers, there is a strong incentive for sources to implement PP + SMART.

In work not shown here for lack of space, we have also simulated a packet-drop strategy in which the arrival of a packet that would overflow the total buffer space triggers the discard of the longest queue, whether or not the arriving packet belongs to that queue. Drop-longest queue is easy to implement in some switch controllers. Under overload, drop-longest-queue give somewhat lower total goodput



(a) Isolated packet losses (b) Bursty packet losses

Figure 6. Tahoe goodput for two random-error scenarios.

than drop-last-packet, but the difference between the two disciplines is not large. Drop-longest-queue is harder on non-Packet-Pair sources than it is on Packet-Pair, because non-Packet-Pair sources keep bumping into the buffer limit and losing their entire queue.

6. TCP Performance with heavy random losses

So far, we have considered losses that occur when many sources contend for a shared buffer (we call this *buffer loss*). Another interesting error model, which is common in wireless networks, is *random loss*. In the baseline simulation, we saw that sources using static window flow control and GBN do not behave well in the presence of buffer loss. The sources do equally poorly in the presence of heavy random losses, because GBN retransmits the entire window on a single loss. Here, we are interested in seeing how well TCP does with random losses.

The behavior of TCP, both Tahoe and Reno versions, under random losses has been extensively studied in [LAKS94]. The following argument displays the important parameter for the small-loss case. As is well known, when there is a single virtual circuit with no losses and no congestion, both versions of TCP approach an ideal steady state in which a uniform stream of packets flows at the speed of the network backbone. If there is an isolated packet loss, TCP cuts its window size and then goes through a two-stage buildup. In the first stage the window size rises quickly to one-half of the round-trip window, and in the second stage the window size increases slowly to the full round-trip window. Suppose we neglect the first stage, and consider only the number of packets transported during the second stage, as compared with the number of packets that would have been transported if there had been no slowdown.

During the second stage, the window increases by one packet after each round-trip time. The effective rate of the virtual circuit during the second stage is $3S/4$, where S is the backbone speed. The duration of the second stage is $(W/2)\,T_{RT}$, where T_{RT} is the round-trip time. The difference between the number of packets carried during the second stage, and the number that would have been carried in the absence of slowdown, is

$$(S - 3S/4)(W/2)\,T_{RT} = ST_{RT}W/8 = W^2/8\,. \quad (4)$$

If the one-way loss rate r is small enough so that loss events do not usually interfere with each other, the effective goodput is approximately

$$\text{Goodput} = 1 - 2rW^2/8 = 1 - rW^2/4\,, \quad (5)$$

assuming that the second term is small compared with unity. Thus, for sufficiently small losses, the fractional reduction in goodput is $rW^2/4$, where W is the number of packets in a round-trip window. Of course the throughput doesn't entirely vanish if $rW^2/4$ is greater than 1, but as Lakshman and Madhow have shown [LAKS94], the performance of TCP with random packet drops is poor if rW^2 is large. By way of illustration, if $W = 100$ packets and the one-way loss rate $r = 1\%$, the parameter $rW^2/4 = 25$.

We confirm this prediction by simulations. We consider a single virtual circuit which has no specific congestion point, but randomly drops packets at some average one-way rate r. Assuming independent losses in each direction, the failure rate for successful acknowledgments is approximately $2r$, if r is small.

We can generalize this model by assuming that packet drops may occur in bursts. Then there are two parameters, namely the average loss rate r and the average length of a loss burst $\overline{L}$. The average length of a good burst is $\overline{G} = (1 - 1/r)\overline{L}$.

Figure 6 shows simulated goodputs for TCP Tahoe with one-way loss rates from 0 to 1% and mean loss bursts of 1 and 5 packets (exponentially distributed burst lengths). In these examples, the goodput is down to a few percent when $r = 1\%$. The results for TCP Reno are similar.

It is clear both from the analysis and from simulations that TCP performance deteriorates rapidly as either the loss rate or the bandwidth-delay product increases. As networks move towards higher speeds, the bandwidth-delay product will inevitably increase. We believe, somewhat counterintuitively, that in future networks, the loss rate too will increase due to three reasons: (a) the presence of wireless links subject to fading, multipath interference, and handoffs, (b) increasing heterogeneity in the network infrastructure, leading to mismatches between link bandwidths and buffer sizes, and (c) as link bandwidths increase, more and more sources will transmit their entire data before receiving any feedback; thus longer-lasting connections are more likely to see losses due to buffer buildups from uncontrolled, short-lived sources. We conclude that there is an an urgent need for a better retransmission strategy than that embodied in TCP. In the next section, we show that, unlike TCP, sources implementing PP+SMART behave nearly optimally in the presence of random losses.

7. Packet-Pair with SMART: Performance with random losses

As we showed in Section 6, for a sufficiently small one-way packet loss rate r, the goodput of a TCP circuit on an otherwise idle network is given approximately by (5), where W is the number of packets in a round-trip window, so that for a wide-area network W^2 can be very large. We now analyze the performance of SMART on a virtual circuit subject to random packet drops (i.e., random losses by some process not necessarily connected with buffer overflow). We will assume that SMART is incorporated into a transport layer protocol that separates error control and flow control (unlike TCP). By this, we mean specifically that the flow control does not slow down when the error control detects a dropped packet, so long as the effective speed of the bottleneck node does not decrease. Only the packets that are actually lost, or whose acknowledgments are lost, have to be retransmitted. Therefore, with such a transport layer, we expect the potential goodput to be as high as

$$\text{Goodput} = (1 - r)^2 \approx 1 - 2r, \qquad (6)$$

if r is the one-way loss rate.

However, as discussed in Section 2, to achieve the maximum goodput, the error-control window may have to be large enough to accommodate several round-trip windows of out-of-sequence packets while lost packets are being retransmitted. We argued in Section 2 that $W_{EC} = 4W_{RT}$ would be sufficient under normal conditions, but larger windows may be required in the presence of heavy losses, or of consecutive-loss bursts. The argument suggests that the limiting goodput (6) can be achieved only as W_{EC} tends to infinity.

To verify our analysis, we simulated SMART on a single lossy link. We used the Packet-Pair flow control protocol as the flow control component of the simulated transport layer. We did not test SMART in TCP because TCP's flow and error control are inextricably linked, and there is no clear way to study the effectiveness of SMART without simultaneously modifying TCP's flow control protocol. Because Packet-Pair has a clean separation of error and flow control, it is a better environment to test retransmission strategies in isolation. Incidentally, this separation is likely to become more common for transport layers layered over ABR service in ATM networks, where the flow control is explicitly managed at the ATM level, and the error control is relegated to the transport level. Thus, our results are directly applicable to ATM transport layers, and have already been implemented in a prototype native-ATM transport layer [AHUJ96].

In the simulations, the offered load is 100%; the source attempts to send continuously at the speed of the bottleneck link. The one-way loss rate varies from 0 to 5%. The mean loss burst size is 1 packet or 5 packets, geometrically distributed.

The router buffer size $B = 1$ round-trip window (100 packets). The results with $B = 0.25$ and $B = 0.50$ are statistically indistinguishable. In the absence of congestion, a single Packet-Pair circuit requires only a very small buffer. The error-control window is equal to 3, 6. and 12 round-trip windows.

Figure 7 shows the results of simulations of Packet-Pair with SMART for losses high enough so that the small-loss approximations do not necessarily hold. (A quick visual comparison of Figs. 6 and 7 may be misleading, because the scales differ on the horizontal axes.) It is clear that SMART + Packet-Pair is much better than TCP; it can achieve nearly the maximum possible goodput under either isolated or bursty losses, provided that the error-control window is several times the round-trip window, in order to allow space for sorting out the retransmissions. It is clear from Fig. 7 that the transport layer's goodput tends toward its limiting value as W_{EC} increases, although the approach is quicker for isolated packet losses (the left side of the figure) than for burst losses (the right side).

8. Discussion

We have presented a new retransmission strategy, SMART, and studied its performance with random line losses as well as buffer losses. The results of our simulation study show that SMART, in conjunction with a rate-based flow control scheme such as Packet-Pair is stable (for all engineering purposes) under congestion, that it can live with a fraction of a round-trip window of buffer space at each queueing point, and that the asymptotic goodput is in the 80-95% range, depending on buffer size. Even if used with naive flow control schemes, SMART can still avert congestion collapse. Our simulations make another important point: a good retransmission strategy alone is not sufficient to deal with packet losses. It must be combined with a good flow control policy. In particular, when Packet-Pair and non-Packet-Pair sources share a congested transmission path, the Packet-Pair sources have an advantage. This suggests that if a vendor advertises per-virtual-circuit queueing and round robin service, self-interest will lead users to use Packet-Pair rate control.

References

[AHUJ96] R. Ahuja, S. Keshav, and H. Saran, ‘‘Design, Implementation, and Performance of a Native-Mode ATM Transport Layer,’’ To appear, Proc. IEEE INFOCOM ’96, (March 1996).

[BONO95] F. Bonomi and K. W. Fendick, ‘‘The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service,’’ IEEE NETWORK, Vol. 9, No, 2 (March/April 1995), pp. 25-39.

[DOSH93] B.T. Doshi, P.K. Johri, A.N. Netravali, and K.K. Sabnani, ‘‘Error and Flow Control Performance of a High Speed Protocol,’’ IEEE Transactions on Communications, Vol. 41, May 1993, pp. 707-720.

[KESH91] S. Keshav, ‘‘A Control-Theoretic Approach to Flow Control,’’ Computer Communication Review, Vol. 21, No. 4 (September 1991), pp. 3-15.

[KESH95] S. Keshav, ‘‘Packet-Pair Flow Control,’’ submitted to IEEE/ACM Transactions on Networking.

[LAKS94] T. V. Lakshman and U. Madhow, ‘‘Performance Analysis of Window-Based Flow Control using TCP/IP: The Effect of High Bandwidth-Delay Products and Random Loss,’’ IFIP Transactions C-26, *High Performance Networking V*, North-Holland (1994), pp. 135-150.

[LAW82] Averill M. Law and W. David Kelton, *Simulation Modeling and Analysis*, (McGraw-Hill, New York, 1982), pp. 145-151.

[SCHW77] M. Schwartz, *Computer-Communication Network Design and Analysis*, Prentice-Hall, Englewood Cliffs, NJ (1977).

[WRIG95] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated*, Vol. 2, Addison-Wesley, Reading, MA (1995).

[ZHAN86] L. Zhang, ‘‘Why TCP Timers Don’t Work Well,’’ Proc. ACM SIGCOMM ’86 (August 1986).

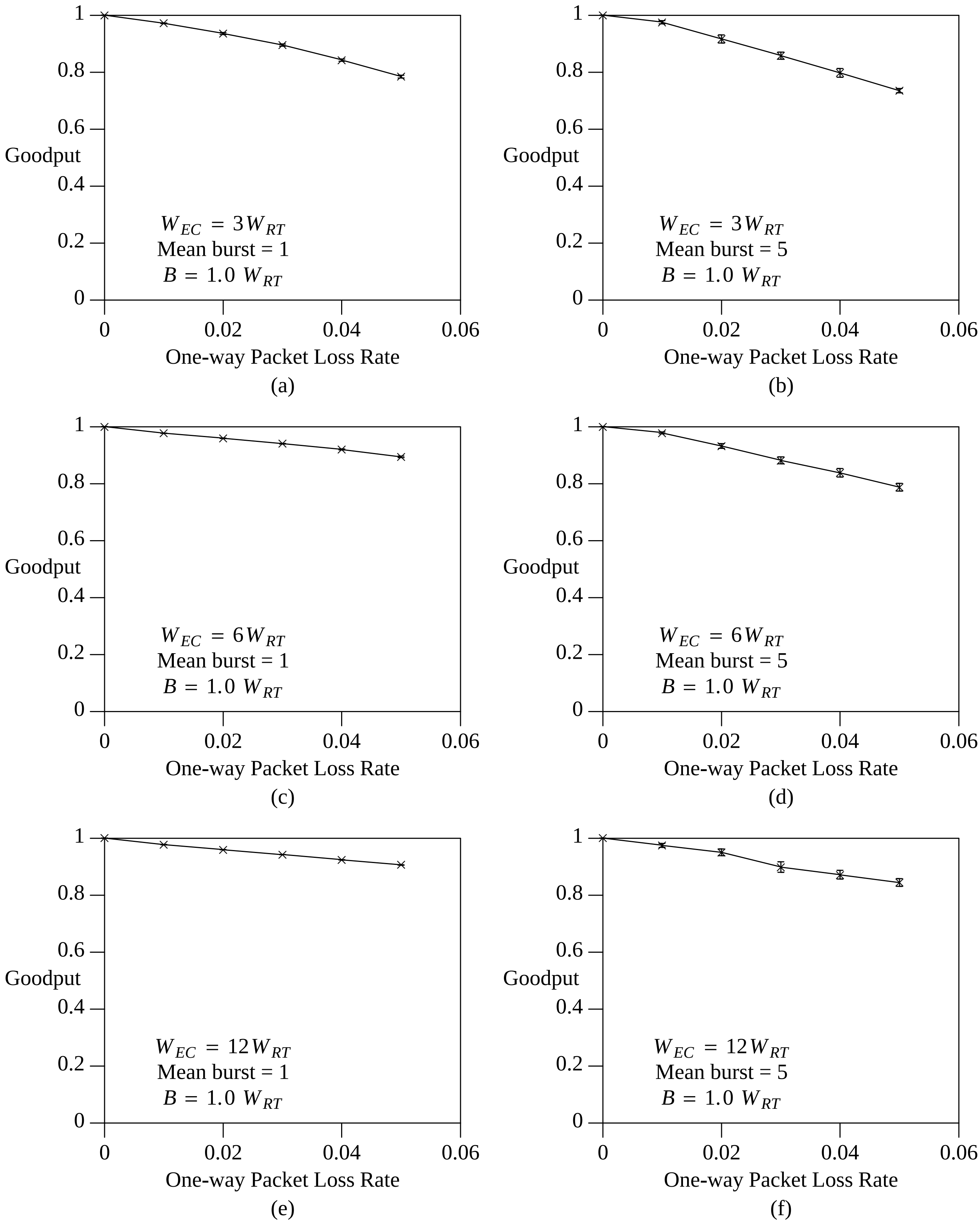


Figure 7. Packet-Pair goodput for isolated and clustered random errors.