Improving Restart of Idle TCP Connections*

USC TR 97-661

Vikram Visweswaraiah John Heidemann

November 10, 1997

Abstract

TCP congestion avoidance mechanisms are based on adjustments to the congestion-window size, triggered by the ACK clock. These mechanisms are not well matched to large but intermittent bursts of traffic, such as responses from a HTTP/1.1-based web server. Idle periods between bursts (web page replies) stop the ACK clock and hence disrupt even data flow. When restarting data flow after an idle period, current implementations either enforce slow start (SSR) or use the prior congestion window (NSSR). The former approach, while conservative, leads to low effective throughput in cases like P-HTTP. The latter case optimistically sends a large burst of back-to-back packets, risking router buffer overflow and subsequent packet loss. This paper proposes a third alternative: pacing some packets at a certain rate until the ACK clock can be restarted. We describe the motivation and implementation of this third alternative and present simulation results which show that it achieves the elapsed-time performance comparable to NSSR and loss behavior of SSR.

1 Introduction

Changes to Internet protocols are often driven by changes in application behavior, thus resulting in different network dynamics and an urge to tune the surrounding protocols for optimal performance. One application that has influenced protocol refinements is the World Wide Web, which uses HTTP running over TCP. The wide use of the Web has emphasized the need to enhance HTTP performance. Persistent-connection support, recently standardized in HTTP/1.1 [6], allows HTTP to re-use a single TCP connection across multiple transactions to the same server [14]. However, the improvements of HTTP/1.1 can interact with surrounding layers. In some cases, we have shown that interactions with TCP can substantially degrade performance [7].

*This research is supported by the Defense Advanced Research Projects Agency (DARPA) through FBI contract #J-FBI-95-185 entitled "Large Scale Active Middleware". The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Army, DARPA, or the U.S. Government. The authors can be contacted at USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, 90292-6695, or by electronic mail to visweswa@isi.edu or johnh@isi.edu.

One of these interactions is the *slow-start restart* problem, originally identified by Jacobson and Karels [12]. Slow-start restart occurs when bursty data is periodically sent over a TCP connection. TCP depends on ACK-clocking for flow control [11]. Idle periods in the connection cause this clocking mechanism to break down. When a burst of data is sent after an idle period, TCP may or may not re-initialize congestion parameters (depending on the TCP implementation) by entering slow-start. If slow-start is initiated, the sender is conservative to the network but incurs delay similar to starting up a new connection (although without the three-way handshake), reducing the benefits of P-HTTP [14]. If the sender does not slow-start then much more data can be quickly sent, but if this burst of data is too large, it can can overrun queues at intermediate routers, leading to packet loss and possibly lower overall performance. A burst of packets can also affect other connections sharing the same router.

The problem with current approaches to transmission in an idle TCP connection is that they either quickly fill the pipe but in a very bursty way (NSSR) or they safely but slowly fill the pipe (SSR). This paper proposes *rate-based pacing* (RBP), an intermediate approach to data transmission after an idle period. RBP paces outgoing packets at a certain rate until the ACK clock is restarted. As shown in Figure 1, RBP attempts to provide a compromise between the extremes of sending back-to-back bursts and restarting with slow start.

This paper demonstrates the problems of current approaches and examines our design for rate-based pacing. We evaluate RBP through simulations and examine our implementation in a controlled laboratory setting.

Although we focus on transmission after an idle period, RBP may be useful other times when many packets are eligible to be sent (for example, when the window jumps for-

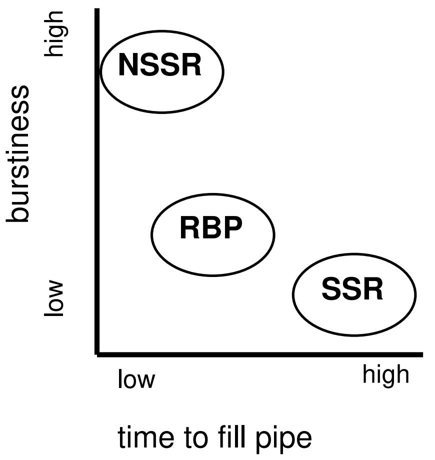

Figure 1: A qualitative comparison of rate-based pacing with slow-start restart and no-slow-start-restart.

ward by a large amount).

2 A Demonstration of the Problem

The behavior of existing TCPs when restarting after a quiescent period can be characterized as either *no slow-start restart* (NSSR) or *slow-start restart* (SSR). For a network operating at heavy load, NSSR has the risk of swamping intermediate routers with bursty data. NSSR can also effectively shut out packets from other streams. This section demonstrates the behavior of TCP with NSSR and SSR on a cross-country internet link, then compares our experimental implementation of pacing.

For this demonstration we sent data between two Sun SPARCstations running SunOS 4.1.3 modified to use either SSR or NSSR. Typical network conditions were 12 hops, 150ms average RTT and approximately 32KB/s bandwidth. Two 512KB chunks of data were transfered between these hosts with a pause of 20 seconds in between. Using `tcpdump` at the sending side, we recorded packets exchanged and graphed TCP sequence numbers against time. The plots focus only on the first 15 segments at the beginning of the second 512KB chunk.

First we consider the case without slow-start restart. By default, SunOS implements NSSR and does not notice idle periods in a TCP connection. This policy can be seen in Figure 2 where a full window of data is transmitted at line rate, appearing as the nearly vertical row of back-to-back packets. In this example, intermediate routers were able to absorb the burst of packets; none were lost. However, a window size larger than 4KB, or independent, nearly concurrent bursts would stress router queueing.

Figure 3 shows the same scenario with slow-start. As can be seen, it takes much longer to return to full transfer rate. After a fairly long stall (due to a delayed ACK for the first segment), there are two or three additional round-trip-length stalls in transfer. This approach is much less bursty than NSSR. Many slow starting connections can proceed concurrently without overloading the network.

These measurements were taken using a simple program sending bulk TCP data. The principles illustrated here extend to HTTP/1.1-style traffic, except for two important TCP implementation issues. First, many TCP implementations (for example, SunOS) do not implement idle-period detection. Second, BSD-derived TCPs approximate an idle connection by time since last data reception. With web traffic, a page request always precedes the data response. This request *resets* idle detection, disabling SSR and causing NSSR-bursts, even though a goal is to avoid bursts in these cases. This bug was first observed by Touch, Oswal, and Hughes and reported on the tcp-impl mailing-list [19].

The overly aggressive nature of NSSR and the conservative nature of SSR suggest an intermediate point. Figure 4 shows behavior one would prefer: a few packets sent at a moderate pace which can restart the ACK clock. Upon receipt of the first ACK, TCP resumes standard ACK-clocked transmission. This example, taken from our prototype RBP implementation, demonstrates the idea behind rate based pacing.

3 Rate-based pacing

Rate-based pacing does not alter basic TCP behavior. Connection set-up and the initial TCP slow-start period remain unchanged. RBP comes into play only after a connection goes idle.

Rather than send a large burst of back-to-back segments (as NSSR) or completely restart congestion control (as SSR), RBP attempts to send out a “reasonable” number of segments which are evenly spaced at a conservative rate. Rate-based pacing ceases as soon as the first acknowledgement is received since the ACK-clock has then been reestablished.

This section clarifies two key design questions for RBP: what is a reasonable number of segments and what is a a conservative rate. We also briefly discuss a few implementation issues.

3.1 A reasonable number of segments

Our goal in choosing a reasonable number of segments to pace is ensuring that RBP can never be worse than NSSR. Using NSSR as the upper bound on RBP assures that RBP won’t be harmful to Internet dynamics, since NSSR is already widely deployed on the Internet (in SunOS, for example).

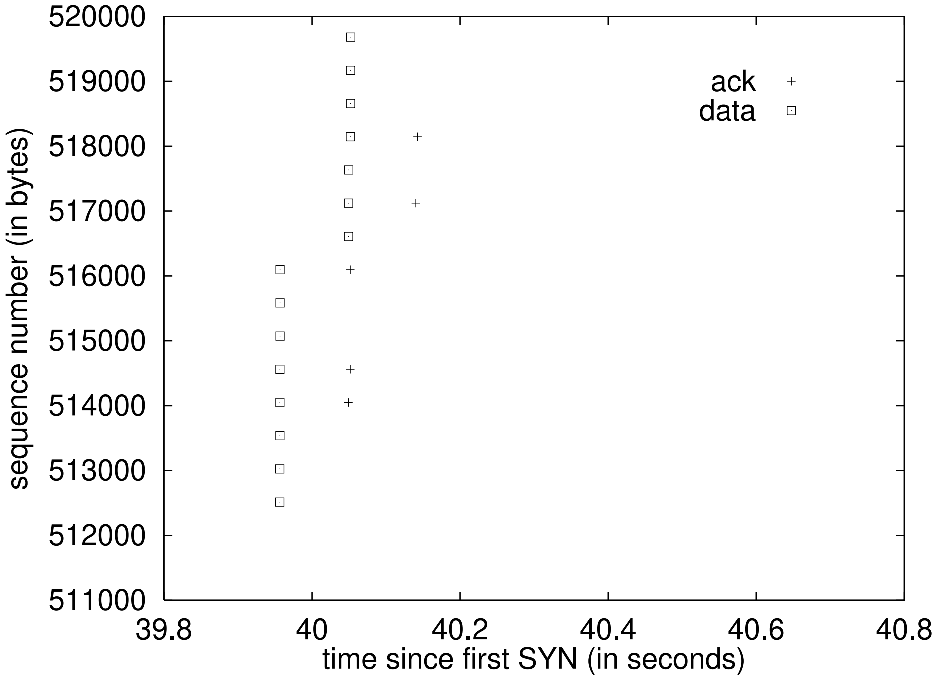

Figure 2: An example of no slow-start restart. The idle period ends at 39.76s with a window-sized, back-to-back burst of data.

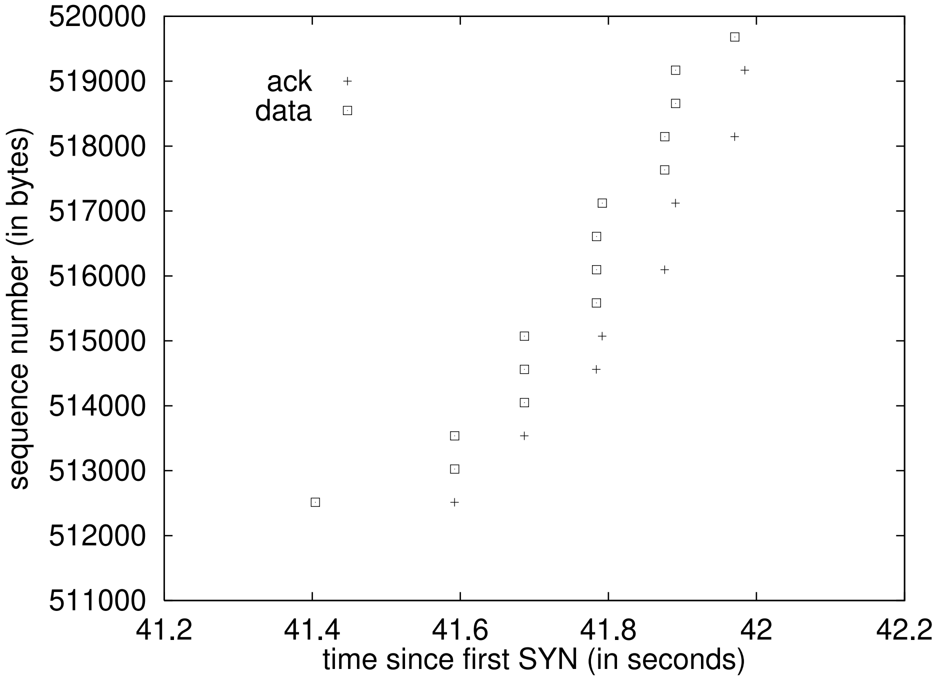

Figure 3: An example of slow-start restart. At 41.4s the idle period ends with a slow start.

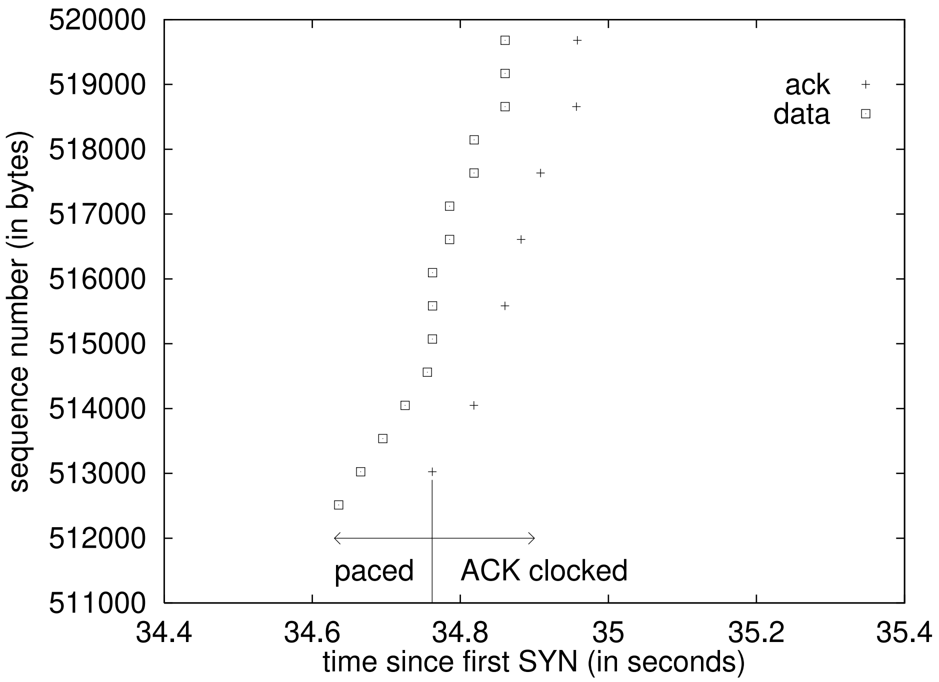

Figure 4: An example of rate-based pacing. The idle period ends with four segments being paced over 34.62–34.76s. Receipt of the ACK at 34.76s causes TCP to resume ACK-clocked transmission.

The minimum number of segments to pace should ensure that the receiver can immediately send an ACK. A host-requirements-compliant receiver must send an acknowledgement after two packets [3], so a lower bound of two segments guarantees an immediate ACK. To avoid being worse than NSSR, the current congestion window forms an upper bound on the number of segments sent. We finally place an upper bound of the number of segments we can send in one estimated round-trip time at the selected sending rate.

3.2 A conservative rate

Without knowing the current amount of network bandwidth *a priori*, the best one can do is use the most recent network observations to estimate available bandwidth and chose a conservative sending rate. It is possible to estimate bandwidth by measuring data sent in a round-trip interval as in TCP Vegas [4].

Recent work suggests that WAN performance is reasonably stable over terms of several minutes [2], but since network conditions may have changed while the connection was idle, the most recently observed rate may no longer hold. To be conservative, we therefore scale this rate downwards by a constant factor. In the future we plan to examine adjusting the scaling factor based on the duration of the idle period. We call this basic rate-selection algorithm *RBP/rate*.

3.3 A less conservative rate

The Vegas algorithm estimates rate by counting segments sent in a round-trip time. This algorithm requires a "full pipe" to accurately measure the rate. For short bursts of data, as are common in web workloads, the Vegas rate estimation algorithm can consistently underestimate available bandwidth.

Since rate estimates can be low, we are experimenting with a more aggressive rate estimate. Our alternative algorithm, *RBP/cwnd*, sends some fraction of the segments that would have been sent in NSSR, but paces them across an entire round-trip time. This approach can be as aggressive as NSSR in the *amount* of data but conservative in the *pace* at which it is sent. Again, the scaling factor can be used to moderate this algorithm.

3.4 Implementation issues

Implementation of RBP/rate requires the following:

1. Idle time detection (to indicate that RBP should be enabled).

2. A method for bandwidth estimation (to compute the

pacing rate).

3. A limitation on how many segments to pace (in case network conditions have dramatically changed).

4. A mechanism to clock (pace) the segments as they are sent.

5. Decision on when to stop RBP.

Idle time detection is done by some existing TCP implementations (4.4 BSD, Linux 2.x). Instead of forcing slow start upon detection of idle time, the behavior is modified to RBP. (Note that RBP is not a substitute for the slow start mechanism; it applies only to the initial phase of data transfer after an idle period.)

We use the USC implementation [1] of TCP Vegas [4] to estimate transfer rate. Currently a scaling factor of 1 is employed (data is sent at the the full prior rate), since for our current workloads the Vegas algorithms consistently underestimate rate. We plan on employing an adaptive algorithm that modifies the scaling factor downwards when large amounts of data are exchanged between idle periods.

We use the existing congestion window to limit the number of segments which can be paced.

Paced segments are clocked by a new timer. Historically, BSD implementations of TCP employ very few timers to minimize timer manipulation overhead. RBP is active only during pacing, minimizing its overhead.

RBP/cwnd does not require a rate estimate and so can be implemented directly over traditional (non-Vegas) TCPs. Our implementation of RBP/cwnd is underway.

4 RBP Evaluation

We can evaluate the effectiveness of rate based pacing by comparing it to SSR and NSSR. We will quantify the effects of these algorithms by comparing elapsed time to data transfered after an idle period and by counting the number of packets dropped due to queue overruns.

4.1 Methodology

We implemented RBP/rate in SunOS, but to easily evaluate RBP over a range of network conditions, this section compares simulations of RBP done in version 2.0b17 of ns [13]. We compare Reno TCP with and without slow-start restart (labeled reno-ssr and reno-nssr in the graphs), and Vegas TCP without slow-start restart (labeled vegas-nssr) against RBP/rate.

The simulations in this paper employ a simplified topology shown in Figure 5. For our simulations there are 10 clients and 1 server. The clients are connected with high-speed links to a single bottleneck link of 800kb/s bandwidth and a 100ms delay. The bottleneck routers have FIFO queueing with a 10KB or 24KB buffer capacity.

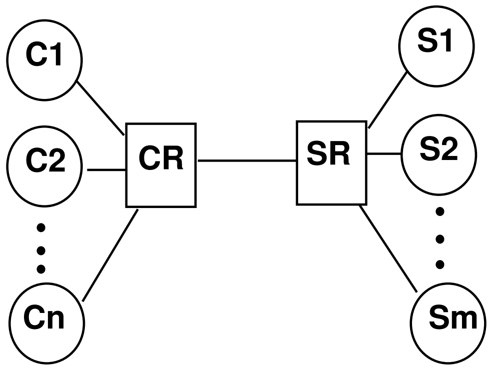

Figure 5: The simulation network configuration: n clients (C1–Cn), and m servers (S1–Sm) connected by a single bottleneck link (CR/SR).

Rather than simulate the entire HTTP protocol, we considered only the responses which consisted of 24KB of data sent from the server to a client. One set of experiments consider variation in this *burst size*. The other set considers variation in the delay of the bottleneck link. An experiment consists of 10 "rounds" of responses. In each round the server replies to each client exactly once, with each reply randomly distributed by ± 1.5s around a central point. Between each round there is 120s of idle time. The server-side router (SR) is then potentially swamped by up to 10 responses every 120s. Since SR's queueing space is limited, replies which are too closely spaced in time may cause packet loss.

4.2 Burst size variation

Our first experiment compares performance as the response size varies. Larger responses increase the chances of router buffer overruns with NSSR and raise the cost of slow-starting with SSR. In this experiment the bottleneck router has a 10KB queue size.

Figure 6 shows how data transfer time changes as we vary response size. We vary the amount of data sent in each reply from 2–40KB, comparing NSSR, SSR, Vegas, and RBP/rate behavior. We can draw two conclusions from this graph. First, for small return sizes (less than about 12KB), NSSR is much faster than SSR or RBP. This is because NSSR is more aggressive at sending data so the sender is stalled fewer times while opening up the congestion window after the idle period. Second, for larger return sizes (more than 12KB), RBP, Vegas and NSSR are about equally fast while SSR is still slower. SSR is slower in this case because of the additional stalls, but RBP is now as fast as NSSR because Vegas' rate-estimation algorithm can do a better job when more data is sent and because for larger bursts NSSR has many more losses than RBP.

Figure 7 shows how packet drop rate changes as burst size varies. As is expected, NSSR and Vegas have a substantial number of dropped packets once individual burst size exceeds the queue size of the bottleneck router. The important conclusion to draw from this graph is that RBP has almost as few packet losses as SSR even at burst sizes of 40KB.

Taken together, we conclude that, as burst size increases, rate-based pacing can offer performance equivalent to NSSR (and better than SSR) while incurring only slightly more packet loss than SSR (and less than NSSR). The difference between RBP and Vegas in Figure 7 also shows the efficacy of pacing as against sending back to back data. By being slightly more aggressive than the very conservative SSR, RBP provides the "best of both worlds" even at bursts of up to 40KB.

4.3 Bottleneck-link delay variation

Examination of RBP performance as burst-size changes suggests that RBP is better than either SSR or NSSR under typical Internet conditions of 100ms latency. Pacing should offer even more promise as round-trip delay rises, such as with satellite links. For this experiment we used a 24KB burst size and bottleneck queue sizes of 24KB (Figures 8 and 9) and 10KB (Figure 10).

Figure 8 shows elapsed time per transaction as the delay of the bottleneck link increases. (Note that we assume symmetric routing while some satellite systems use an asymetric back-channel.) Larger round-trip delay increases all elapsed times, but this increase is disproportionately large for the SSR case, since there are substantially more stalls than the other algorithms and each stall incurs a round-trip-time penalty [8]. Under the conditions of our experiment, we conclude that RBP and NSSR behave similarly as delay changes.

Figure 9 compares loss as delay changes. As can be seen, NSSR, RBP and Vegas have higher losses than SSR, which has very little loss. Since both burst size and router queue size are 24KB, packet loss should only occur when two responses are very closely spaced in time. Under typical Internet conditions, the size of the bottleneck queue is not under our control and so there is no guarantee that burst size is less than queue size. To see the effect of this situation, we reduced the bottleneck queue size from 24KB to 10KB and reran the experiment (see Figure 10). Under these conditions, NSSR and Vegas both have a substantial loss rate (about 40% of packets need to be retransmitted). Both SSR and RBP instead show relatively little packet loss.

Again, these experiments suggest that RBP can offer NSSR-like performance with SSR-like loss rates. The performance advantages of RBP are larger as RTT increases, while its better loss characteristics are most important when router queue size cannot be adjusted to match traffic patterns.

5 Related Work

Rate-based pacing builds on several areas of related work. It builds directly on Jacobson's description of ACK clocking [11] and Jacobson and Karel's proposal of slow-start after an idle period [12]. Several others have also observed that "less (transmission) is more (throughput)" in the context of TCP. Nagle showed that delayed transmission of small packets can be advantageous [15]. More recently, Hoe showed that a limiting slow-start can improve performance [10]. Hoe's work is complementary to ours; she improves slow-start behavior while we offer an alternative when more is known about network conditions. Hoe also suggested (independent of our work), pacing as a potential alternative start up mechanism for TCP [9].

Rate-limited algorithms have been widely explored in the literature, beginning with leaky bucket [20]. NETBLT explored a rate-based alternative to TCP [5].

Finally, rate-based pacing assumes that the the prior observed rate has some predictive value for current rates (possibly with a conservative scaling factor). Balakrishnan et al. argue that network conditions are relatively stable for periods of several minutes [2].

6 Future Work

Our experiments suggest that rate-based pacing can offer NSSR-like performance and SSR-like loss rates for a range of network conditions. Although RBP is promising, additional work is required to more fully understand RBP behavior, and several issues must be considered for RBP deployment.

6.1 Additional evaluation

Several areas of RBP would benefit from additional evaluation. Preliminary experiments suggest that RBP/rate and RBP/cwnd behave similarly. We plan to examine these issues more closely.

Our current experiments evaluate performance where all traffic is running the same algorithm. We would like to compare a mix of algorithms and evaluate RBP fairness in this context.

To examine a wide range of network characteristics, this paper considers RBP simulations. We would like to validate our simulations against our implementation of RBP/rate.

Finally, RBP applies old estimates of network bandwidth

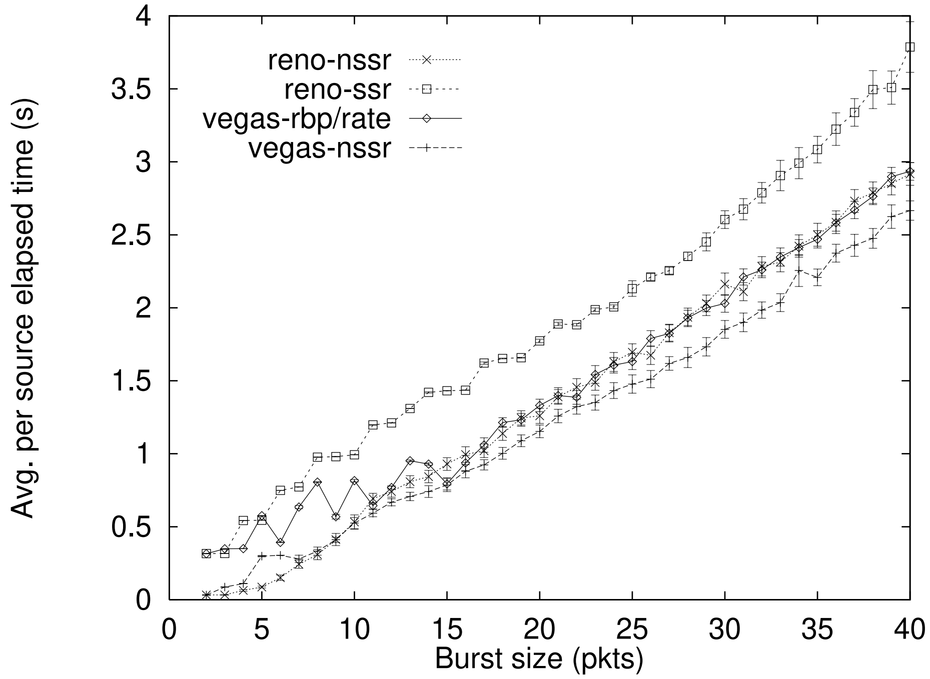

Figure 6: Average elapsed time per transfer as reply size varies. Each data point is the mean of 10 simulations with 10 clients per simulation. Bars show 90% confidence intervals.

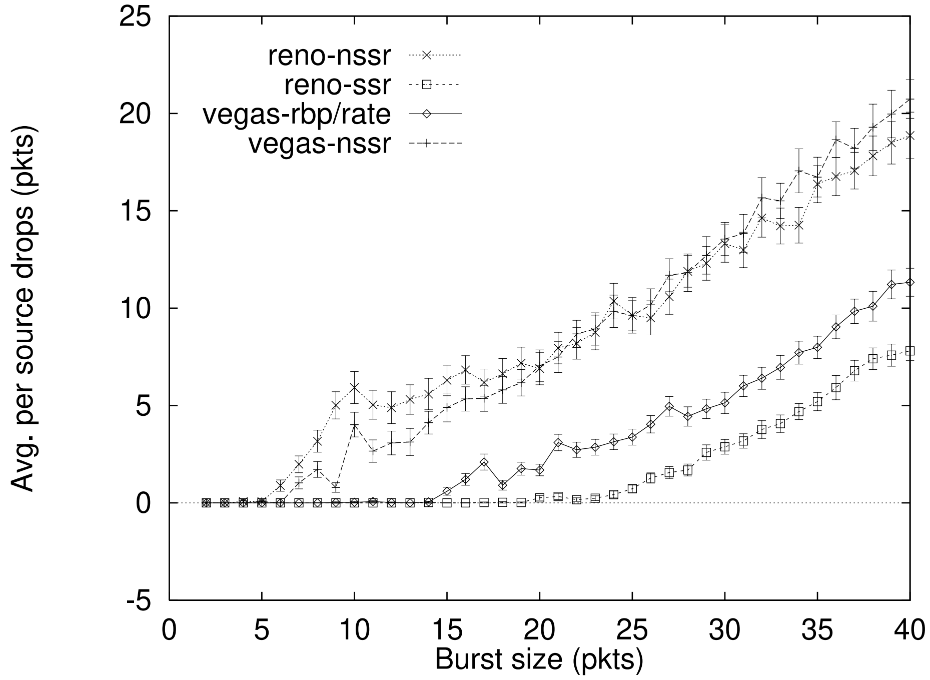

Figure 7: Average packet drop rate per transfer as reply size varies. Each data point is the mean of 10 simulations with 10 clients per simulation. Bars show 90% confidence intervals.

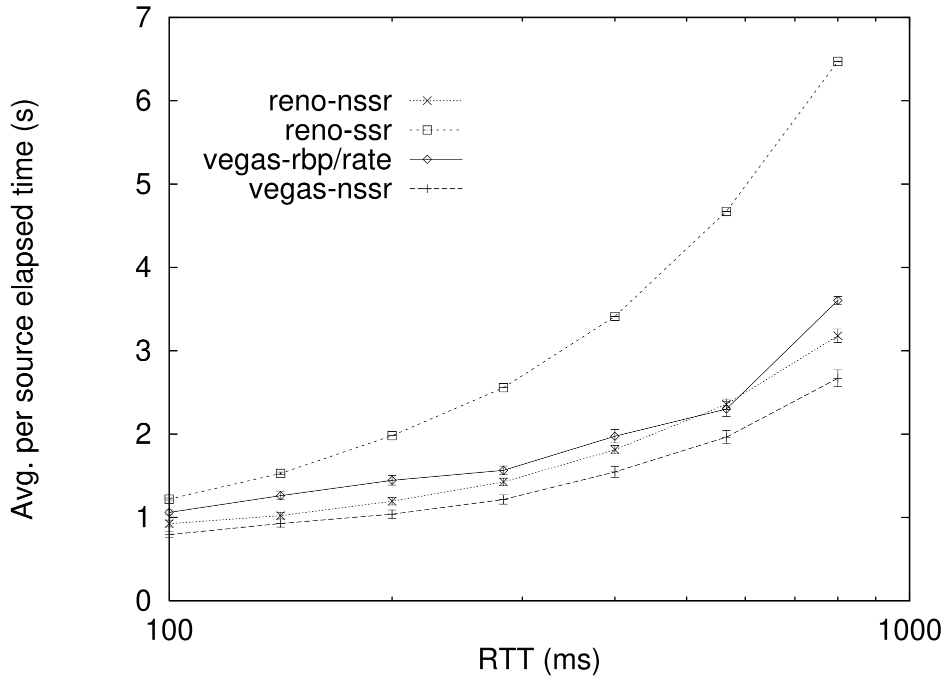

Figure 8: Average elapsed time per transfer as bottleneck-link delay varies. Each data point is the mean of 10 simulations with 10 clients per simulation. Bars show 90% confidence intervals.

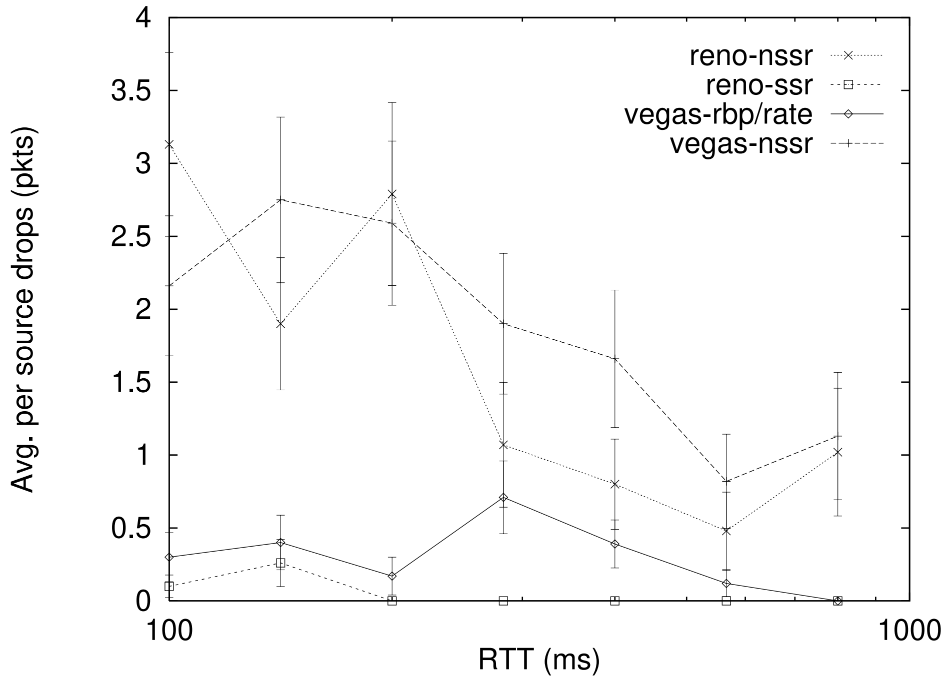

Figure 9: Average number of packet losses per transfer as bottleneck-link delay varies. Each data point is the mean of 10 simulations with 10 clients per simulation. Bars show 90% confidence intervals.

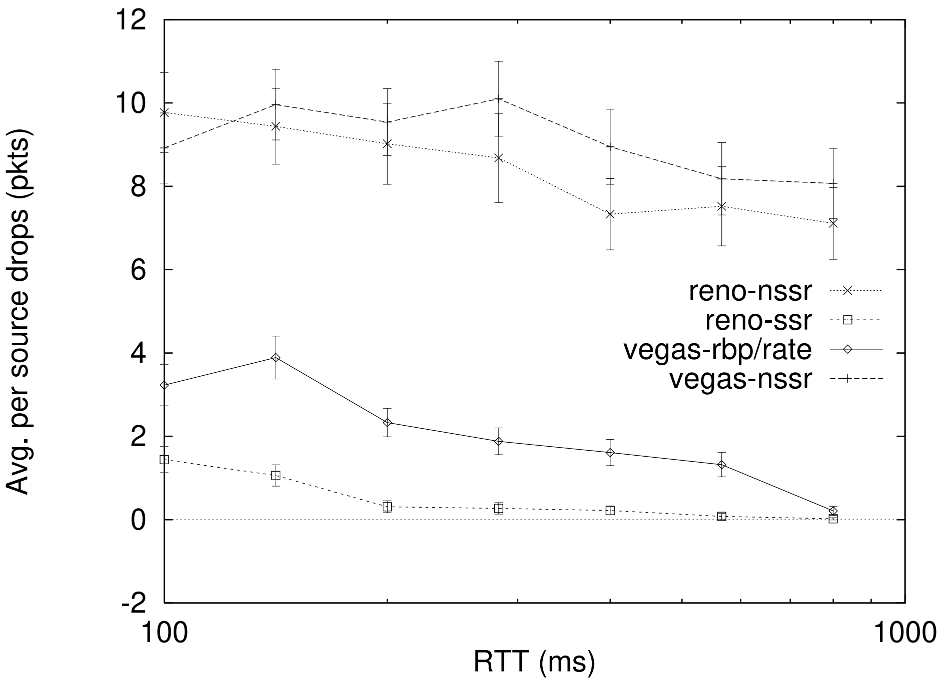

Figure 10: Average number of packet losses per transfer as bottleneck-link delay varies. Each data point is the mean of 10 simulations with 10 clients per simulation. Bars show 90% confidence intervals.

to current conditions. We are experimenting with more general approaches to caching and sharing network conditions across both time (sequential connections) and space (concurrent connections) [18].

6.2 Deployment issues

Several aspects of RBP simplify deployment.

RBP requires only sender-side modifications to TCP. This makes it particularly beneficial for HTTP/1.1-based web servers, thus providing motivation for web-server maintainers to incrementally deploy RBP.

Our current implementation of RBP/rate makes use of Vegas' rate estimate. RBP/cwnd does not require a rate estimate and so can be used in traditional TCP implementations.

6.3 Broader applications

We have applied pacing to the problem of data transmission after an idle period. Pacing solves the more general problem of controlled transmission of many TCP segments when the ACK-clocking mechanism breaks down.

Idle connections are an example of sender-induced breakdown of the ACK-clock. Similar problems can also occur because of problems with the receiver or the network. A slow receiver may buffer and consume data in increments of a full window. When information that the window has been consumed reaches the sender, a full window of data can be sent immediately. Improper TCP receiver implementations can also result in lack of ACK clocking (for example, Solaris 2.5 as described by Paxson [16]).

Packet loss in the network can also derail ACK clocking. A single segment early in the window may be lost but the full window successfully received. When the single lost segment is retransmitted and received, the window will jump forward. Fast retransmit is one approach to restart the ACK clock in this case [17]; pacing could also be used.

TCP-over-satellite is another case where pacing may be useful. Because satellites have a large pipe to fill, slow start is particularly expensive. If network characteristics were somehow known (either through outside knowledge or possibly through information cached from prior connections) it may be desirable to use pacing when starting new connections.

7 Conclusions

This paper has described rate-based pacing, a new approach to restart mid-stream data transfer that is congestion sensitive and not unnecessarily slow. RBP allows TCP to quickly reestablish TCP's ACK clock after the connection goes idle. By pacing traffic based on an estimate of network performance, RBP can be more aggressive than a complete slow-start (SSR) but can avoid the burstiness of back-to-back packet transmission (NSSR).

We have implemented RBP in SunOS and examined its performance through simulation. We have shown that RBP provides the performance of NSSR and the loss characteristics of SSR for a range of network conditions. Protocols such as HTTP/1.1 use TCP in a bursty manner, where fetches of a web page (text and images) are separated by long idle periods. Rate-based pacing will improve performance for this kind of traffic.

References

[1] J.S. Ahn, Peter B. Danzig, Z. Liu, and L. Yan. TCP Vegas: Emulation and experiment. In *Proceedings of the ACM SIGCOMM '95*, pages 185–195, Cambridge, Massachusetts, August 1995. ACM.

[2] Hari Balakrishnan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. Analyzing stability in wide-area network performance. In *Proceedings of the ACM SIGMETRICS*, Seattle WA, USA, June 1997. ACM.

[3] R. Braden. Requirements for Internet hosts—communication layers. RFC 1122, Internet Request For Comments, October 1989.

[4] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the ACM SIGCOMM '93*, San Francisco, CA, September 1993. ACM.

[5] David D. Clark, Mark L. Lambert, and Lixia Zhang. NETBLT: A high througput transport protocol. In *Proceedings of the ACM SIGCOMM '87*, pages 353–359. ACM, August 1987.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol—HTTP/1.1. RFC 2068, Internet Request For Comments, January 1997.

[7] John Heidemann. Performance interactions between P-HTTP and TCP implementations. *ACM Computer Communication Review*, 27(2):65–73, April 1997.

[8] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the performance of HTTP over several transport protocols. To appear, IEEE/ACM Transactions on Networking, November 1996.

[9] Janey C. Hoe. Start-up dynamics of TCP's congestion control and avoidance schemes. Master's thesis, Massachusetts Institute of Technology, May 1995.

[10] Janey C. Hoe. Improving the start-up behavior of a congestion control scheme for tcp. In *Proceedings of the ACM SIGCOMM '96*, pages 270–280, Stanford, CA, August 1996. ACM.

[11] Van Jacobson. Congestion avoidance and control. In *Proceedings of the SIGCOMM '88*, pages 314–329. ACM, August 1988.

[12] Van Jacobson and Mike Karels. Congestion avoidance and control. *ACM Computer Communication Review*, 18(4):314–329, August 1990. Revised version of his SIGCOMM '88 paper with an additional appendix.

[13] Steve McCanne, Sally Floyd, and Kevin Fall. Ucb/lbl network simulator ns. at http://www-mash.cs.berkeley.edu/ns/ns.html, 1996.

[14] Jeffrey C. Mogul. The case for persistent-connection HTTP. In *Proceedings of the SIGCOMM '95*, pages 299–313. ACM, August 1995.

[15] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, Internet Request For Comments, January 1984.

[16] Vern Paxson. Automated packet trace analysis of tcp implementations. In *Proceedings of the ACM SIGCOMM '97*. ACM, 1997.

[17] W. Richard Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Work in progress (Internet draft draft-stevens-tcpca-spec-01.txt), March 1996.

[18] Joe Touch. TCP control block interdependence. RFC 2140, Internet Request For Comments, April 1997.

[19] Joe Touch, Anand Oswal, and Amy Hughes. Does this qualify as a bug? (the failure of bsd to ssr for web traffic). Message 199708121739.KAA08900@rum.isi.edu on the mailing-list tcp-impl@engr.sgi.com, August 1997.

[20] Jonathan S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(4):8–14, October 1986.

Acknowledgments

We would like to thank Ted Faber and Joe Touch for comments about the paper.

Software Availability

Our implementation of RBP/rate for SunOS-4.1.3 is currently available at the LSAM web pages at ⟨`http://www.isi.edu/lsam/`⟩. We plan to make an RBP/cwnd implementation available shortly. Simulated versions of RBP/cwnd and RBP/rate are available in the 2.0b17 release of ns at ⟨`http://www-mash.cs.berkeley.edu/ns/`⟩.