Proceedings of the 2nd Annual Los Alamos Computer Science Institute Symposium (LACSI 2001).

Packet Spacing: An Enabling Mechanism for Delivering Multimedia Content in Computational Grids*

Annette C. Feng†‡, Wu-chun Feng§, Geneva Belford†
{afeng, feng}@lanl.gov, belford@cs.uiuc.edu

† Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Ave.
Urbana, IL 61801

‡ Computing, Communications, and Networking Division
§ Computer & Computational Sciences Division
Los Alamos National Laboratory
P.O. Box 1663
Los Alamos, NM 87545

Abstract

Streaming multimedia with UDP has become increasingly popular over distributed systems like the Internet. Scientific applications that stream multimedia include remote computational steering of visualization data and video-on-demand teleconferencing over the Access Grid. However, UDP does not possess a self-regulating, congestion-control mechanism; and most best-effort traffic is served by congestion-controlled TCP. Consequently, UDP steals bandwidth from TCP such that TCP flows starve for network resources. With the volume of Internet traffic continuing to increase, the perpetuation of UDP-based streaming will cause the Internet to collapse as it did in the mid-1980's due to the use of non-congestion-controlled TCP.

To address this problem, we introduce the counter-intuitive notion of inter-packet spacing with control feedback to enable UDP-based applications to perform well in the next-generation Internet and computational grids. When compared with traditional UDP-based streaming, we illustrate that our approach can reduce packet loss over 50% without adversely affecting delivered throughput.

Keywords*: network protocol, multimedia, packet spacing, streaming, TCP, UDP, rate-adjusting congestion control, computational grid, Access Grid.*

*This work was supported by the U.S. Dept. of Energy through Los Alamos National Laboratory contract W-7405-ENG-36. This paper is LA-UR 01-0904.

1 Introduction

TCP and UDP are the most widely-used transport protocols today, the TCP/IP protocol suite being the *de facto* standard in the Internet-computing environment. TCP enables reliable, bulk-data transfer; however, it is inappropriate for such tasks as live video-on-demand and remote computational steering of visualization data in computational grids. Bulk-data transfer requires 100% reliable communication, and hence, TCP. Video-on-demand and remote computational steering generally do not require 100% reliability, therefore, TCP is overkill. For instance, if a video frame is missing a small block of pixels due to a lost packet, the video application is better off displaying the virtually complete frame and moving on to the next frame instead of waiting for the re-transmission of the lost packet (which over the Internet could easily take 100 ms).[1] TCP, in this case, provides too much functionality because its loss detection and re-transmission mechanisms, being tightly integrated with TCP's congestion-control mechanism, are inherent functions of the protocol.

UDP, on the other hand, provides no reliability guarantees. Specifically, it provides best-effort, end-to-end service without performing loss detection and packet re-transmission and without performing congestion control. Because of this, UDP obtains more bandwidth than TCP,

[1] If the required frame rate is 30 frames per second, then the interframe delay is only 33 ms. Therfore, a re-transmission delay of 100 ms over the wide-area network is clearly unacceptable.

albeit at the risk of suffering packet loss and packet reordering, problems that can ultimately be addressed by the applications themselves. Therefore, multimedia applications such as RealPlayer [17, 18] and scientific applications such as remote data visualization use UDP in order to improve perceived performance. Because UDP does not self-regulate in response to network congestion, these UDP-based applications gobble up available network resources, stealing bandwidth away from well-behaved applications that use congestion-controlled TCP. An application that blasts UDP packets into the network can readily fill the buffers of an intermediate router, causing severe congestion and packet loss. Since TCP-based applications slow down their sending rates in response to congestion, these applications become starved for network resources as the UDP-based applications continue to blast their packets unchecked into the network and claim the bandwidth being made available to them. Even though sending hosts can inject UDP packets as quickly as they are able, the throughput can suffer dramatically due to heavy packet loss and increased delays as packets spend more time waiting in queues within the network.

A simple observation reveals that adequate throughput can be attained by spacing the packets apart instead of blasting them one right after the other into the network. The next section reveals this insight. The notion of slowing down the sending rate in order to achieve better throughput is certainly counter-intuitive; however, our experiments show the viability and effectiveness of this approach.

1.1 Insight

Based on our recent work in network traffic characterization [25, 8, 9], we observed significant packet loss even when the offered load was less than half of the available network bandwidth. An analysis of our *ns* [1] simulations revealed that this behavior was due to simultaneous bursts of traffic coming from client applications and overflowing the buffer space in the bottleneck router. Metaphorically, this could be viewed as what happens at a major highway interchange during rush hour where everyone wants to go home simultaneously at 5:00 p.m., thus “overflowing” the highway interchange. To avoid such a situation, some people self-regulate themselves by heading home at a different time, i.e., spacing themselves out from other people.

If we view vehicles as packets and the highway interchange as a router, then to avoid buffer overflow and enhance throughput, packets should not be blasted onto the network one after another. Instead, packets should be spaced out over time. To test this hypothesis, we ran live wide-area network (WAN) tests between Los Alamos National Laboratory (LANL), University of Illinois at Urbana-Champaign (UIUC), and Ohio State University (OSU). These tests consisted of sending UDP packets between LANL and either UIUC or OSU at different packet-spacing intervals. Figures 1 and 2 show the throughput and packet loss, respectively, of a representative test between LANL and UIUC [6]. When the packet spacing is zero, e.g., today’s UDP-based multimedia-streaming applications, the throughput is 62 Mb/s but with a packet loss of almost 90%! With as little as 100 μs of spacing between packets, the throughput remains the same, but the packet loss drops all the way down to 35%. And when the packet spacing is 50 μs, the throughput is actually *higher* than when the packets are not spaced as in UDP-based multimedia streaming.

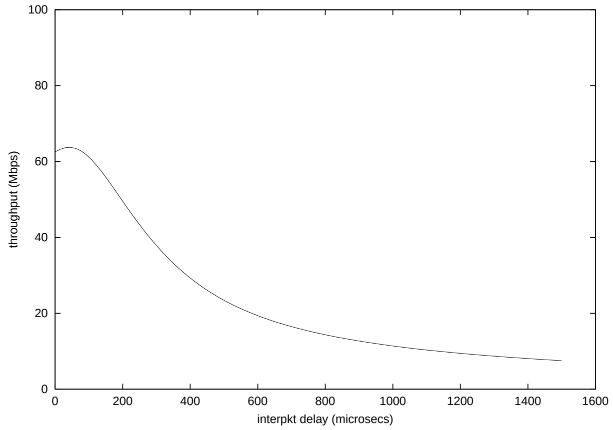

Figure 1. Delivered Throughput to the Receiver

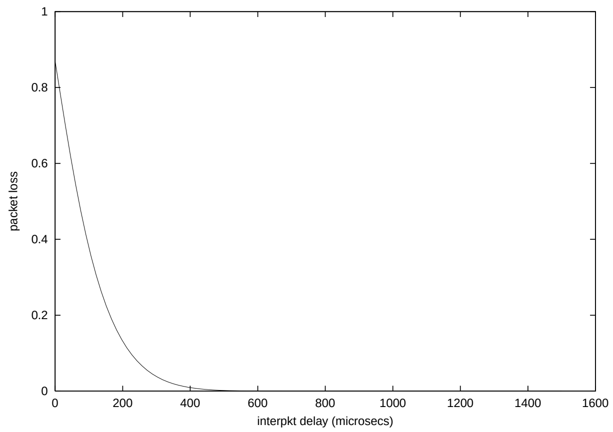

Figure 2. Packet-Loss Percentage

All curves from our other live WAN tests have the same general shape. That is, the throughput initially increases when the amount of packet spacing increases and then decreases exponentially as the amount of spacing increases further. The packet-loss percentage immediately decreases in an exponential manner as packet spacing increases.

1.2 Related Work

Many transport protocols for the delivery of multimedia content certainly have been proposed, among them being XTP, RAP, and HPF. The Xpress Transport Protocol (XTP) [4] uses explicit rate control to combat congestion, however, the congestion-control mechanism must be implemented within the network and not simply at the edges. Fluctuating round-trip times (RTTs) cause poor performance because of a design feature whereby XTP enters a synchronizing handshake when a timer expires while XTP awaits a response to a request for information on missing data [3]. Furthermore, because it is a complex protocol, XTP is meant to be implemented in VLSI for performance reasons, so software implementations are too slow for multimedia traffic [23].

The Rate Adaptation Protocol (RAP) [20] is a TCP-friendly protocol that employs an "additive increase, multiplicative decrease" (AIMD) algorithm for rate adjustment. RAP is intended for the transmission of delay-sensitive, semi-reliable, rate-based applications which use layered-encoding of their data streams. RAP is therefore not a general solution but specifically targets layered-encoded multimedia content which it uses to adjust its transmission rate by adjusting the number of layers it sends.

The Heterogeneous Packet Flows (HPF) [13] protocol supports the delivery of packets having differing QoS requirements within a single stream. Addressing a design flaw of TCP, HPF decouples congestion control from reliability and uses a rate-based, AIMD approach to combat congestion. The problem with the AIMD approach (also used by RAP) is that such an approach will not scale to high-performance (or more precisely, high bandwidth-delay product) networks. For example, when the window size is one, a linear increase is a 100% increase. When the window size is 1000, a linear increase is a mere 0.1%. An absolute linear increase in window size from 500 to 1000 (as during TCP's congestion-avoidance phase) will take 500 round-trip times to converge! More realistically, the situation is actually much worse. If we assume a typical WAN with a high bandwidth-delay product, i.e., 1 Gb/s WAN $\times$ 100 ms RTT = 100 Mb, then for an uncongested network, the ubiquitously deployed TCP reno continually increases its window size until it induces packet loss (i.e., just after 100 Mb) and then chops its window size in half (i.e., 50 Mb). The reconvergence back to the "optimal window size" of 100 Mb using TCP's absolute linear increase takes much too long and results in lowered network utilization. In this particular case, convergence can take as long as (100 Mb - 50 Mb) / (1500 B/RTT * 8 b/B) = 4,168 RTTs or (4,168 RTTs * 100 ms/RTT) = 416.8 seconds = 6.947 minutes!

In 1997, Mahdavi and Floyd [14] informally proposed the notion of equation-based congestion control for unicast applications. While the AIMD algorithm found in TCP backs off by cutting its sending rate in half in response to a single congestion indication, equation-based congestion control uses a control equation that more gradually and smoothly adapts its maximum rate because some real-time applications find that halving the sending rate is unnecessarily severe and can noticeably reduce the user-perceived quality [24]. Although the above work has given rise to a significant amount of research on equation-based and other types of congestion-control mechanisms [22, 20, 24, 16, 21, 10], we still do not have any deployable congestion-control mechanisms for best-effort streaming multimedia.

Previous work in packet spacing includes [12, 2]. In [12], Jain argues that rate-control protocols for congestion control may not work without the cooperation of intermediate routers because packets may get clumped together at the intermediate routers anyway. This would result in larger bursts at the intermediate routers even though the goal may have been to reduce the burstiness of the traffic. While this may have been true a decade ago, we believe that the boom of the world-wide web and other multimedia applications creates enough interleaving traffic to maintain packet spacing between end hosts. We will substantiate this belief in Section 3.2.4.

Aggarwal et al. [2] study the effect of uniform packet spacing (or "pacing") over a round-trip time in TCP. While pacing results in better fairness, throughput, and lower drop rates in some cases, the throughput is worse than regular TCP most of the time because a paced-TCP is susceptible to synchronized losses and delays congestion notification. In contrast, we focus on the effects of packet spacing over UDP with control feedback rather than on TCP itself.

In general, our packet-spacing protocol differs from the above work in several ways. First, rather than focusing primarily on being compatible or *fair* with TCP, our rate-adjusting protocol addresses fairness while simultaneously delivering UDP-like bandwidth. Second, we accomplish the above feat by introducing the counterintuitive notion of packet spacing. Third, rather than relying on equation-based congestion control to more smoothly adapt the sending rate, we allow the sending rate to adapt as needed (based on available network resources). We then rely on transcoding, e.g., mapping a multimedia stream onto rapidly-varying available bandwidth [19], to smooth out any potentially rapid change in available bandwidth.

2 Approach

Packet spacing refers to the delay introduced between two consecutive packets, as shown in Figure 3. Here, t_s is the amount of spacing between packets, and t_x is the transmission time for each packet. By introducing such a delay, bursts of packets can be spaced out, resulting in fewer packet drops at intermediate routers and potentially *higher* throughput at the end host, as shown back in Figure 1. Thus, packet spacing can potentially be used as a mechanism to assist in congestion avoidance and control.

Based on Figure 1, the ideal operating region of our packet-spacing mechanism ranges from 50 μs to 500 μs. No packet spacing or packet spacing of less than 50 μs results in very high packet loss with less delivered bandwidth than when the packet spacing is 50 μs.

Depending on the application, the ideal packet-spacing range may be as small as 100 μs to 200 μs in order to get UDP-like bandwidth but with significantly less packet loss, e.g., at 200 μs, bandwidth is 50 Mb/s while packet loss is only 10%, or as large as 400 μs to 500 μs to obtain TCP-like reliability but with higher throughput than TCP. To exploit this counterintuitive finding, we develop an ad-hoc packet-spacing protocol (PSP) to adjust the amount of packet spacing based on feedback from the network.[2]

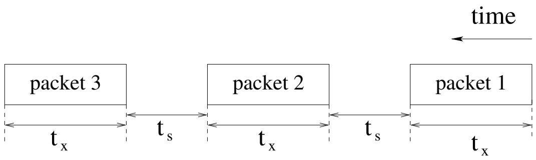

Figure 3. Packet Spacing

2.1 Ad-Hoc Packet-Spacing Protocol (PSP)

In our ad-hoc packet-spacing protocol (PSP),[3] the sender initially transmits packets at the highest possible rate, i.e., no inter-packet spacing, and the receiver sends acknowledgments every round-trip time (RTT) for the packets it receives. (This RTT is the base propagation-delay time, not the dynamic RTT. To keep the protocol simple, we did not experiment with dynamic RTTs.)

We calculate the base RTT by performing *ping* during connection set-up.[4] After the connection is established, the sender conveys the calculated RTT to the receiver by including it within the header of each packet. Note that this is not required after the first acknowledgment is received, but we have left this provision so that dynamic RTTs can be used in the future. Each acknowledgment contains the number of packets that were received in the previous RTT.

When the sender receives such acknowledgments, it compares the number of packets sent, p_{sent}, in the previous RTT to the number of packets received, p_{rcvd}. Based on the values of p_{sent} and p_{rcvd}, the sender adapts its packet spacing ps as shown in Figure 4.

if $p_{sent} > p_{rcvd}$ (i.e., packets were lost) **then**
    /* sender must reduce its transmission rate */
    if $ps = 0$ **then**
        $ps \leftarrow 50\, \mu s$
    else
        $ps \leftarrow min(ps * 2, RTT)$
else /* sender tries to increase its sending rate */
    $ps \leftarrow ps - 2$

Figure 4. Ad-Hoc Packet-Spacing Protocol

Because our WAN experiments and simulations showed that the ideal packet spacing occurred between 0 μs and 2000 μs, we chose an initial packet spacing of 50 μs because (1) anything smaller generated significantly higher packet loss with no benefit with respect to throughput and (2) finding the ideal packet spacing within this range quickly would take no more than seven RTTs. Larger spacings can be reached in only a few more RTTs because the packet spacing increases exponentially.

The $ps \leftarrow min(ps * 2, RTT)$ clause ensures that the maximum packet spacing is one RTT. This ensures that at least one packet is sent every RTT.

2.2 Damped Packet-Spacing Protocol

Due to the opposing packet-spacing decisions in PSP, our initial tests of PSP resulted in large oscillations around the ideal sending rate. To address this problem, we added the following heuristic to damp the oscillations: *If a loss occurs due to a deliberate decrease in the packet spacing (and consequently, increase in rate), then the sender reverts to the previous packet-spacing value.* Using this heuristic, the sender makes significantly smaller oscillations around the ideal operating point. Figure 5 shows a comparison between the PSP and damped PSP. In this figure, each experiment ran for 100 s, and the sending rate for each was plotted. With damping, the overall throughput increased by 10%.

3 Experiments

For our WAN simulations, we used *ns-2*, which is a network simulator developed by the VINT group [1]. We will refer to senders and receivers as *agents*, which follows naturally from the terminology employed by *ns-2*. Our simple

[2] We note that at the present time, the feedback is only used for adjusting the packet spacing and that no retransmissions are done at this time.

[3] We refer to our protocol as being "ad-hoc" because it is an ad-hoc, point-specific solution meant to illustrate the benefits of packet spacing. It is not a general solution (although we are currently in the midsts of testing a general solution).

[4] A more sophisticated mechanism could be developed to get a better estimate of the RTT. However, for the purposes of our experiments, we only needed a value that was reasonable enough to provide timely feedback.

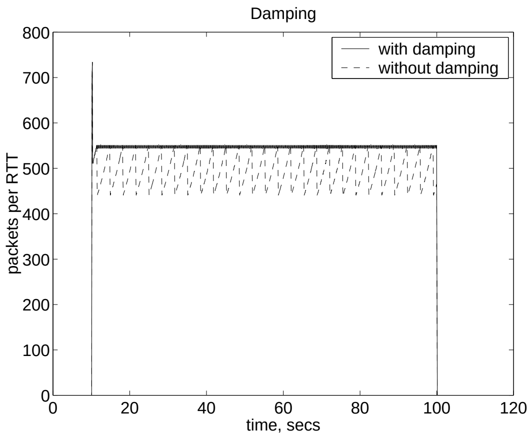

Figure 5. Oscillation Damping

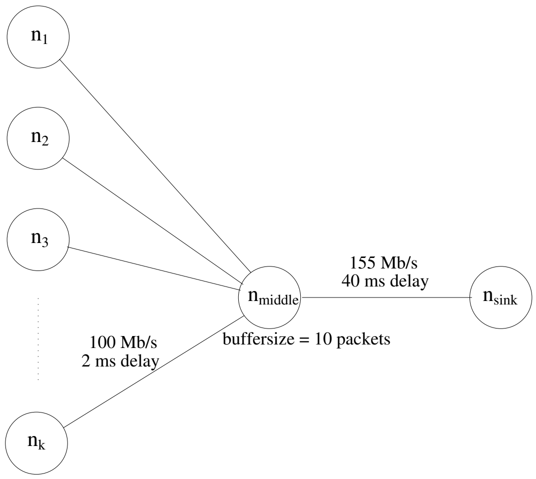

Figure 6. Topology for WAN Simulations

packet-spacing agents (PSAs) implement packet spacing without feedback while our adaptive packet-spacing agents implement the damped PSP rather than the simple PSP.

3.1 Network Topology

Figure 6 shows the network topology that we used in our experiments. The k nodes on the left $(n_1, n_2, \ldots, n_k)$ simulate senders on a local-area Ethernet, transmitting via a common gateway router (e.g., LAN/WAN gateway or n_{middle}) to a WAN backbone running at 155 Mb/s or OC-3; this topology models the LAN and WAN at Los Alamos National Laboratory. All the receivers are aggregated into the node n_{sink}. The gateway router has a buffer size of 10 packets, 100-Mb/s Ethernet links with 2-ms delays to the senders, and a 155-Mb/s link with 40-ms delay to the receivers. This delay is typical of the delay found in a transcontinental WAN connection.

3.2 PSA Simulations

Here we study the behavior of (1) a single PSA with no other traffic, (2) competing PSAs, and (3) PSAs competing with TCP agents. Like Mo et al. [15] who compare TCP Reno and TCP Vegas using infinite file transfers, we use infinite file transfers for the TCP connections as well. (For the figures in this section, each data point in the simulation graphs represents the result of a 500-s simulation for a particular packet-spacing interval.)

3.2.1 Single PSA

Figure 7 shows the throughput for a single PSA for packet spacings between 0 μs and 5000 μs. (Note that there is no other traffic on the network besides that of the single PSA.) As expected, the sender and receiver throughputs are the same. This is because the gateway can keep up with the aggregate sending rate and because there is no competing traffic on the channel, and therefore, no packet loss.

3.2.2 Competing PSAs

In this set of experiments, we ran simulations with 2, 4, 8, and 16 PSAs competing against each other, respectively. Figures 8 and 9 show the results for the last case. The resulting behavior is similar to what we observed in the actual WAN experiments (i.e., Figures 1 and 2). (Note that all the 16 competing PSAs showed similar behavior.)

In Figures 8 and 9, the region of interest occurs between 0 μs and 1000 μs. With a packet spacing of 0 μs, the sender throughput is 100 Mb/s while the receiver-realized throughput is only a measly 10 Mb/s with a packet loss of 90%! As packet spacing increases, the packet-loss percentage drops sharply, and the throughput at the receiver actually *increases* to its maximum point at 1000 μs of inter-packet spacing. This phenomenon is similar to what we found with our live WAN tests in Figures 1 and 2.

3.2.3 PSAs Competing with TCP Agents

In these experiments, we ran simulations with 1, 2, 4, 8, and 16 sender/receiver TCP pairs and an equal number of PSA pairs, respectively. Figures 10 and 11 show the behavior of one particular PSA competing with 15 other PSAs and 16 TCP connections. All other simulations resulted in similar behavior. Again, we see that the behavior is strikingly similar to that seen in the actual WAN experiments. The optimal performance of the PSAs with respect to throughput

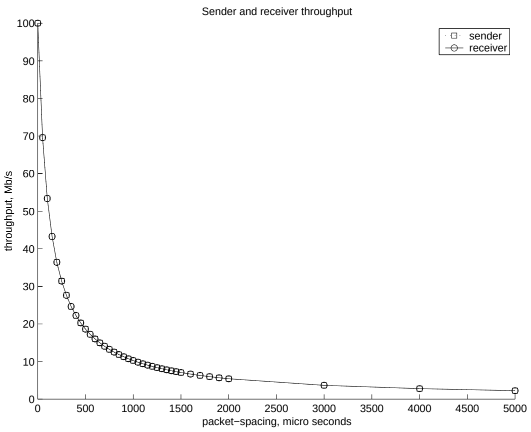

Figure 7. Throughput for One PSA

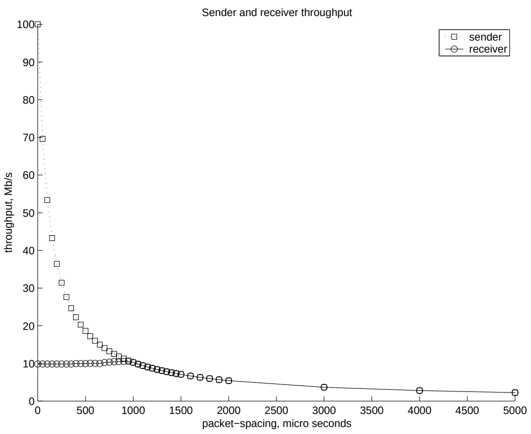

Figure 8. Throughput for One of the 16 PSAs

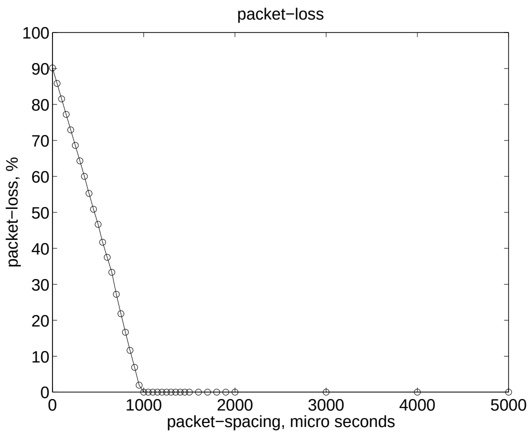

Figure 9. Packet Loss for One of 16 PSAs

and packet loss occurs at 1000 μs to 1050 μs, i.e., throughput is 11 Mb/s while packet loss is 0%.

Figures 12 and 13 show the throughput and packet-loss behavior of one particular TCP connection competing with 15 other TCPs and 16 PSAs, respectively. In these figures, we cannot help but notice that the TCP throughput does not increase beyond 2.7 Mb/s (even when the PSA throughput is low)! The reason for this behavior has nothing to do with the TCP-friendliness of our damped PSP and has everything to do with TCP's default advertised receiver window of 20 packets. This receiver's window size is the default in many operating systems and artificially limits the amount of outstanding data that a sender can have in the network. Further, the figures also show that with very small packet spacings, the PSAs operate like UDP connections (as expected), thus starving TCP connections of any bandwidth.

Figures 14 and 15 show how TCP behaves with a window large enough to keep a bandwidth-delay product's worth of information outstanding in the network. These figures show that with sufficient spacing by the PSAs, a TCP connection can consume its share of available bandwidth. For example, Figures 10 and 14 illustrate that with 5000 μs of packet spacing, each PSA receiver sees 2.23 Mb/s while each TCP receiver gets 6.57 Mb/s.

3.2.4 PSA Spacing at the Receiver

To verify our claim that packet-spaced traffic stays spaced out by the time it reaches the receiver (rather than getting clumped as claimed by [12]), we recorded the inter-arrival time of packets at one PSA receiver, using the same experimental set-up as described in Section 3.2.3. The sending PSAs used a spacing of 1500 μs; the resulting inter-packet spacings at the receiver averaged 1540.6 μs with a standard deviation of 64.75 μs.

3.3 Adaptive PSA Simulations

Our adaptive PSAs implement the damped PSP, which tries to find the ideal packet spacing under varying network conditions. We first show the behavior of two adaptive PSAs competing against each other and then with two additional TCP connections. As in Section 3.2, the TCP connections were that of infinite file transfers.

3.3.1 Competing Adaptive PSAs

Figure 16 shows how the sending rate of an adaptive PSA varies with time. The adaptive PSA makes small oscillations around the ideal sending rate. Figure 17 demonstrates that our adaptive PSAs are fair (when both are started simultaneously) as both adaptive PSAs have sending rates that lie on the fairness line.

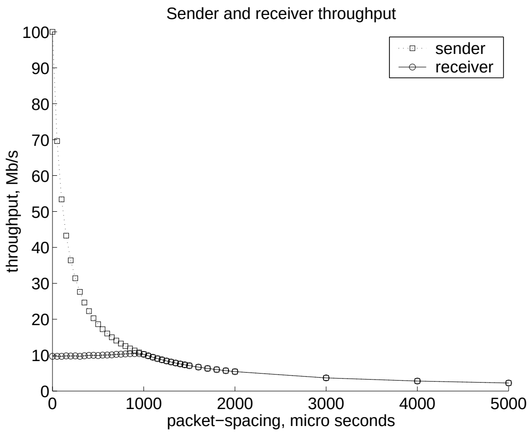

Figure 10. Throughput for One PSA of 16 PSAs and 16 TCP Connections

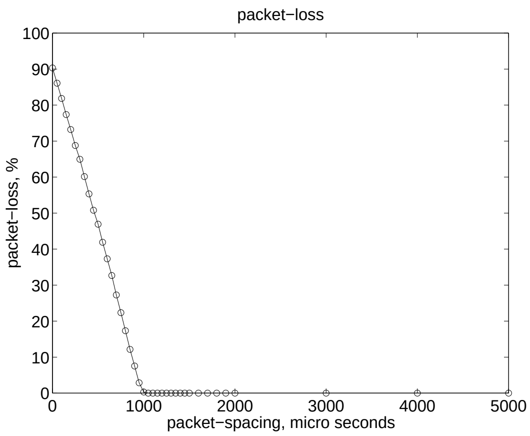

Figure 11. Packet Loss for One PSA of 16 PSAs and 16 TCP Connections

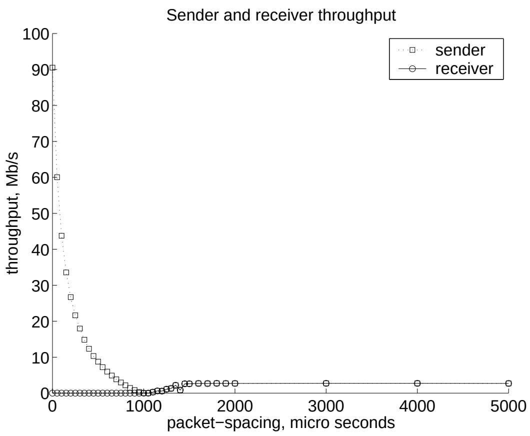

Figure 12. Throughput for One TCP of 16 TCP Connections and 16 PSAs

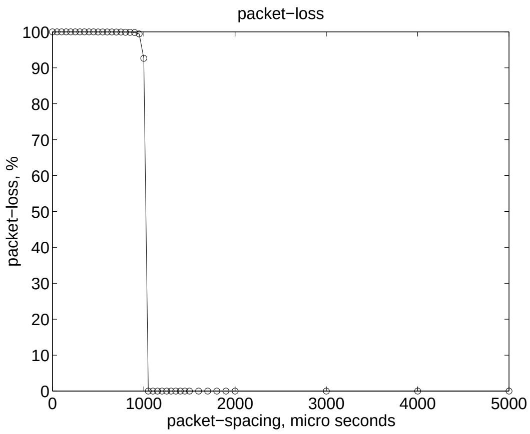

Figure 13. Packet Loss for One TCP of 16 TCP Connections and 16 PSAs

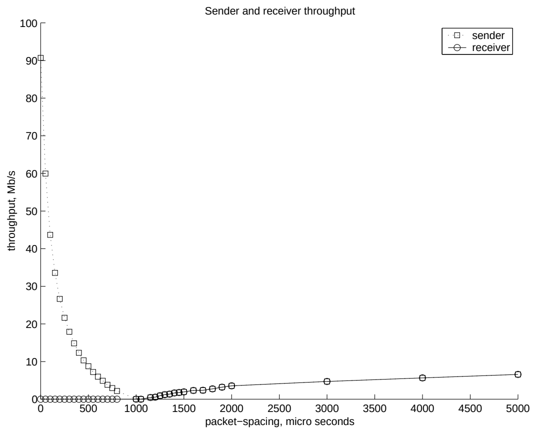

Figure 14. Throughput for One TCP (Window Size = 800 Packets) of 16 TCP Connections and 16 PSAs

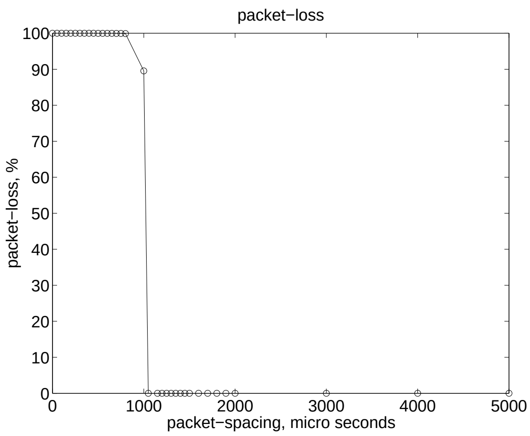

Figure 15. Packet Loss for One TCP (Window Size = 800 Packets) of 16 TCP Connections and 16 PSAs

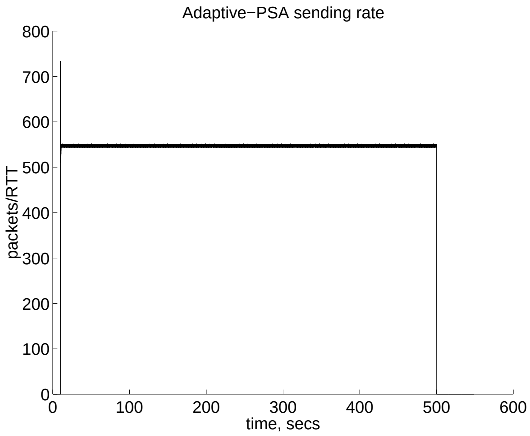

Figure 16. Sending Rate for One of Two Adaptive PSAs

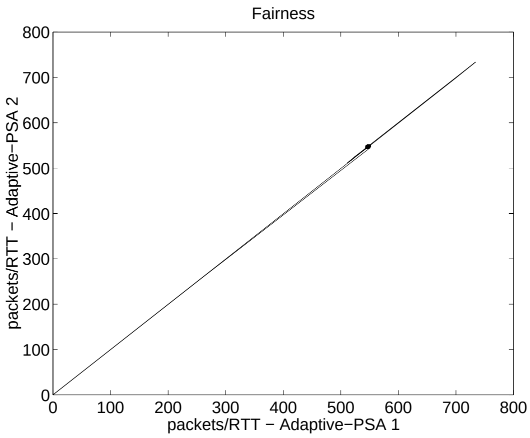

Figure 17. Fairness - Simultaneous Start

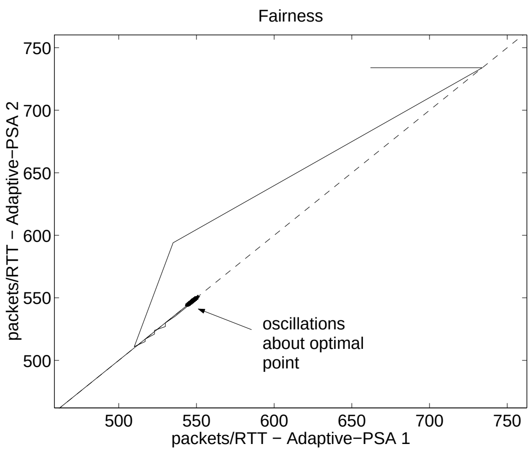

Figure 18. Fairness - Delayed Start

Figure 18 shows us a portion of the fairness graph where one adaptive PSA started 10 seconds later than the other. As we can see, both adaptive PSAs change their rates in a fair manner and eventually make small oscillations about the ideal sending rate.

3.3.2 Competing Adaptive PSAs with Background Traffic

In this simulation, we ran 10 TCP connections with infinite file transfers in the background and two adaptive PSAs competing in the foreground. Figure 19 shows that the adaptive PSAs respond readily to congestion. And again, both adaptive PSAs have very similar sending rates.

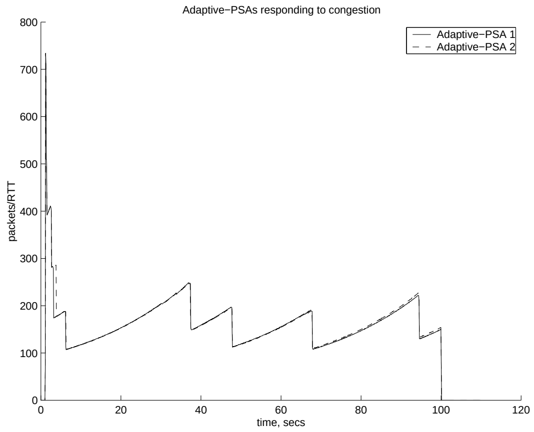

Figure 19. Two Adaptive PSAs Competing with 10 TCPs

4 Implications for Next-Generation Internet

The results in Section 3 support our claim that "packet spacing" is preferable to "packet blasting" because of reduced packet loss, increased throughput, and increased fairness. The packet-spacing protocol is a solution that works for today's Internet and for tomorrow's next-generation Internet, which will introduce smart routers with active queue management [11, 7]. These routers will punish packet-blasting UDP applications by dropping packets from their non-adaptive flows.

Many applications do not need the full reliability of TCP, and hence, should not use TCP as their transport mechanism, e.g., video teleconferencing in the Access Grid [5]. UDP is the main alternative. It does not provide any reliability guarantees, but neither does it provide for, much less enforce, congestion control. As a result, UDP-based applications, currently stealing bandwidth that results from its lack of congestion control, will be crippled further as smart routers with active queue management make their way into the next-generation Internet infrastructure. The purpose of incorporating these smart routers into the Internet is two-fold: (1) to allow routers to enforce the implicit, defacto, fair-usage policies as they have evolved in the best-effort Internet and (2) to reduce queue lengths within the network so that the network is better able to absorb the natural packet bursts that occur in normal network traffic.

Smart routers employing active queue management schemes in the next-generation Internet will have some measure of control over "rogue" applications to ensure that they do not unfairly steal bandwidth away from competing applications and fill up all the available buffer space within the network. In light of this coming reality, streaming applications must have a viable alternative to TCP and UDP with respect to flexibility (in terms of reliability) while providing adequate and fair congestion control to be "good" network citizens. Our damped packet-spacing protocol is a first step in providing such an alternative.

5 Conclusion

Perhaps the most interesting result in this paper is that a receiver's realizable throughput actually *increases* (up to a point) even when the sender's transmission rate decreases. This result has dramatic implications on many of today's multimedia applications that blast packets onto the network as fast as possible, i.e., no packet spacing. By slowing down the introduction of packets into the network, congestion is alleviated at the intermediate routers; this, in turn, results in a net increase in throughput. Thus, this work provides an incentive for multimedia provides not to blast UDP packets indiscriminately into the network. In addition, it provides motivation for the deployment of a packet-spaced protocol that can deliver UDP-like performance yet still be responsive to competing connections, particularly for applications with multimedia streaming such as the Access Grid [5].

Our damped packet-spacing protocol (PSP), implemented via an adaptive PSA, sends data near its "optimal" sending rate by using a simple feedback mechanism that reports packet loss every RTT. This mechanism in turn controls the amount of packet spacing. Our preliminary results demonstrate that by introducing packet spacing to a multimedia stream, packet loss can be reduced dramatically without much loss in throughput.

Future work includes examining the performance of our damped PSP with different types of application traffic and over a live WAN. Of particular interest are those applications that generate data in short bursts with relatively large intervals between bursts. Based on the experimental results presented here, we expect that the packet loss that would normally be induced by these bursts to be greatly reduced.

References

[1] UCB/LBNL/VINT network simulator - ns (version 2). http://www.isi.edu/nsnam/vint/index.html.

[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In *Proceedings of IEEE INFOCOM*, 2000.

[3] J. W. Atwood and G. C. K. Chung. Error Control in the Xpress Transfer Protocol. In *Proceedings of 18th Conference on Local Computer Networks*, pages 423–431, September 1993.

[4] Y. Baguette and A. Danthine. Comparison of TP4, TCP and XTP - Part 2: Data Transfer Mechanisms. *ETT*, 3(5), September-October 1992.

[5] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi. Access Grid: Immersive Group-to-Group Collaborative Visualization. In *Proceedings of the 4th International Immersive Projection Technology Workshop*, 2000.

[6] A. Feng. RAPID: Rate-Adjusting Protocol for Internet Delivery. Thesis Proposal (in progress), University of Illinois at Urbana-Champaign, 2001.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. In *Proceedings of IEEE INFOCOM*, April 2001.

[8] W. Feng and P. Tinnakornsrisuphap. The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems. In *Proceedings of ICPP'00*, August 2000.

[9] W. Feng and P. Tinnakornsrisuphap. The Failure of TCP in Distributed Computational Grids. In *Proceedings of SC'00*, November 2000.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. of SIGCOMM 2000*, August 2000.

[11] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[12] R. Jain. Myths about Congestion Management in High Speed Networks. *Internetworking: Research and Experience*, 29(2):101–113, February 1992.

[13] J. Li, S. Ha, and V. Bharghavan. HPF: A transport protocol for supporting heregeneous packet flows in the internet. March 1999.

[14] J. Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control. Technical report, Note sent to end2end-interest mailing list, January 1997.

[15] J. Mo, J. Walrand, and V. Anantharam. Analysis and Comparison of TCP Reno and Vegas. In *Proceedings of IEEE INFOCOM*, 1999.

[16] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A Model Based TCP-Friendly Rate Control Protocol. In *Proceedings of NOSSDAV'99*, 1999.

[17] RealNetworks Inc. RBN$^{\mathrm{TM}}$ (Real Broadcast Network) White Paper. Available at *http://www.real.com/solutions/-rbn/whitepaper.html*, January 1999.

[18] RealNetworks Inc. RealVideo Technical White Paper. Available at *http://www.real.com/devzone/library/whitepapers/-overview.html*, January 1999.

[19] R. Rejaie, D. Estrin, and M. Handley. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proceedings of ACM SIGCOMM '99*, September 1999.

[20] R. Rejaie, M. Handley, and D. Estrin. An End-to-End Rate-Based Congestion Control Mechanism for Real-Time Streams in the Internet. In *Proceedings of INFOCOM'99*, March 1999.

[21] I. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP Emulation at Receivers — Flow Control for Multimedia Streaming. Technical report, North Carolina State University, April 2000.

[22] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme. In *Proceedings of NOSSDAV'98*, 1998.

[23] W. T. Strayer, S. Gray, and R. E. C. Jr. An Object-Oriented Implementation of the Xpress Transfer Protocol. In *Multimedia: Advanced Teleservices and High-Speed Communication Architectures, 2nd International Workshop (IWACA'94)*, pages 387–400, September 1994.

[24] D. Tan and A. Zakhor. Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol. *IEEE Transactions on Multimedia*, May 1999.

[25] P. Tinnakornsrisuphap, W. Feng, and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. In *Proceedings of ICDCS'00*, April 2000.