Journal of Supercomputing, Vol. 23, No. 1, August 2002.

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

Annette C. Feng (afeng@lanl.gov)
Apu C. Kapadia (akapadia@cs.uiuc.edu)
Los Alamos National Laboratory
Los Alamos, NM 87545
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Wu-chun Feng (feng@lanl.gov)
Los Alamos National Laboratory
Los Alamos, NM 87545
Department of Computer & Information Science
The Ohio State University
Columbus, OH 43210

Geneva G. Belford (belford@cs.uiuc.edu)
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

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 01-4904.

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, albeit at the risk of suffering packet loss and packet re-ordering, problems that can ultimately be addressed by the applications themselves. Therefore, multimedia applications such as RealPlayer [19, 20] 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.

[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.

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 [8,9,26], 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.

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.

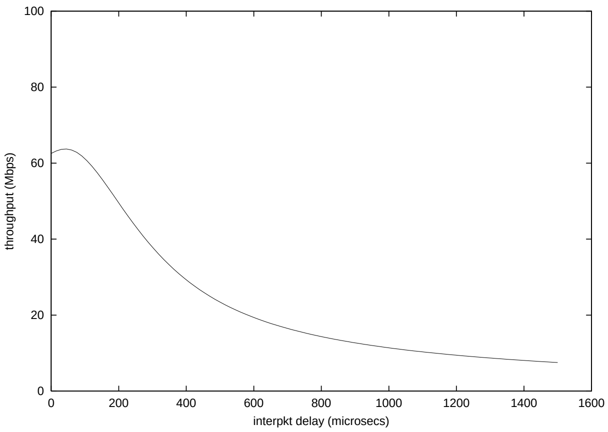

Figure 1. Delivered Throughput to the Receiver

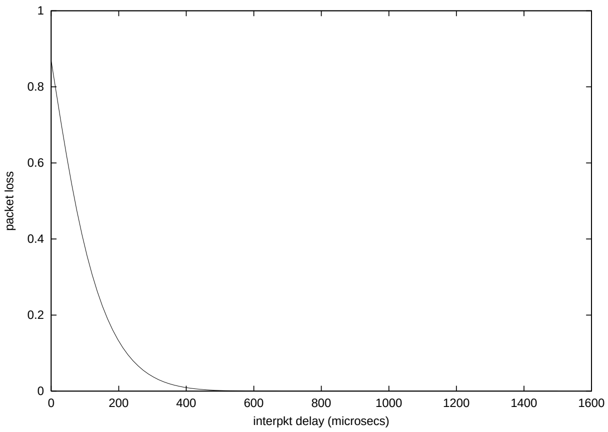

Figure 2. Packet-Loss Percentage

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 [24].

The Rate Adaptation Protocol (RAP) [21] 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) [14] 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 re-convergence 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 [15] 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 [25]. Although the above work has given rise to a significant amount of research on equation-based and other types of congestion-control mechanisms [11, 18, 21–23, 25], we still do not have any deployable congestion-control mechanisms for best-effort streaming multimedia.

Previous work in packet spacing includes [2,13]. In [13], 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 3.2.3.

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.

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]

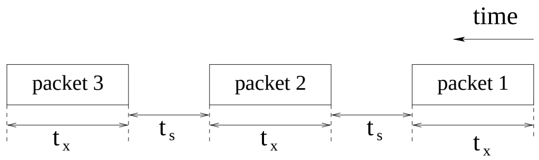

Figure 3. Packet Spacing

2.1. Ad-Hoc Packet-Spacing Protocol

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.

Because our WAN experiments and simulations showed that the ideal packet spacing occurred between 0 μs and 2000 μs, we chose

[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 this protocol as being "ad-hoc" because it is a point-specific solution that only applies to the tested topology and serves only to demonstrate the benefits of packet spacing. It is not 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.

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

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. That is, at least one packet is sent every RTT.

2.2. Damped Packet-Spacing Protocol

Due to the opposing packet-spacing decisions in our ad-hoc PSP, 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, resulting in a 10% increase in overall throughput. Thus, in this paper, we will present experimental results for the damped PSP rather than the ad-hoc PSP.

2.3. Equation-Based Packet-Spacing Protocol

Although the purpose of the ad-hoc and damped PSPs is to demonstrate the benefits of packet spacing, these protocols are point-specific solutions that apply only to the tested topology shown in Figure 5. Furthermore, the halving of the sending rate in the ad-hoc and damped PSPs in response to a single congestion indication may be unnecessarily severe for real-time applications such as video and induces bursty traffic behavior. To address these problems, we discuss a general solution that adapts an equation-based, congestion-control mechanism [11] and incorporates packet spacing.

An application that uses a congestion-control mechanism that is more aggressive than TCP's could unfairly limit TCP traffic from getting its fair share of bandwidth [10] and vice versa. To achieve a proper balance, we select a formulation that is based on the steady-state sending rate of TCP [17]:

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

This equation provides an upper bound on the sending rate T in bytes/sec, as a function of the packet size s, round-trip time R, steady-state loss event rate p, and the TCP retransmit timeout t_{RTO}.

While setting the rate according to the above equation ensures TCP-friendliness between the equation-based packet-spacing protocol and TCP, it will result in bursty traffic behavior because TCP itself is bursty in nature. Thus, we set the sending rate not only based on the above equation but also on an exponentially weighted moving average of the number of packets between loss events [11] in order to smooth out changes in the sending rate.

3. Experiments

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

Our packet-spacing agent (PSA) implements packet spacing *without* any congestion-control feedback. Because the damped PSP outperforms the ad-hoc PSP, we only implement and experiment with the former in a damped PSA. Lastly, our equation-based PSA implements the equation-based PSP.

3.1. Network Topology

Figure 5 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.

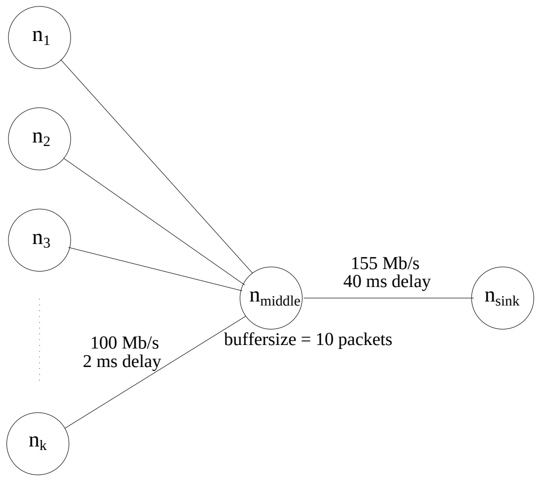

Figure 5. Topology for WAN Simulations

3.2. PSA Simulations

Here we study the behavior of competing PSAs and PSAs competing with TCP agents. Like Mo et al. [16] 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. *PSAs Competing*

In this set of experiments, we ran simulations with 2, 4, 8, and 16 PSAs competing against each other, respectively. Figures 6 and 7 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 6 and 7, 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.

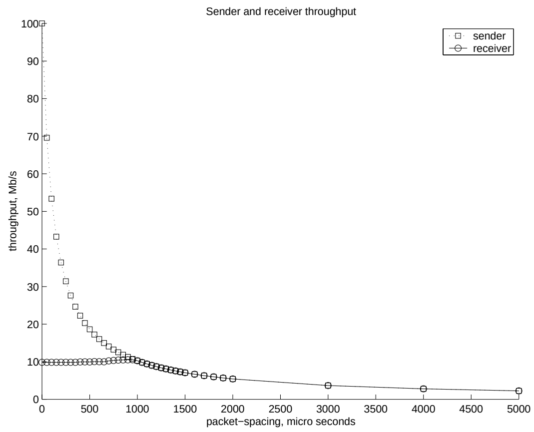

Figure 6. Throughput for One of the 16 PSAs

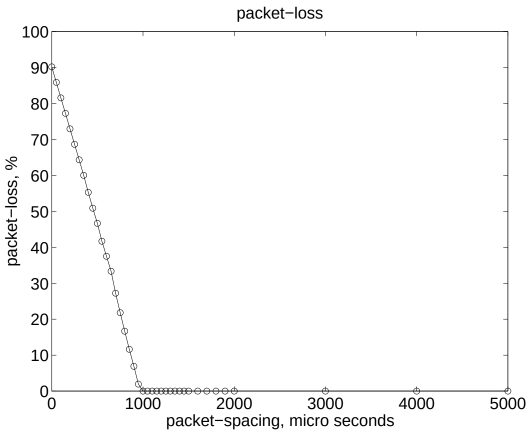

Figure 7. Packet Loss for One of 16 PSAs

3.2.2. *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 8 and 9 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 and packet loss occurs at 1000 μs to 1050 μs, i.e., throughput is 11 Mb/s while packet loss is 0%.

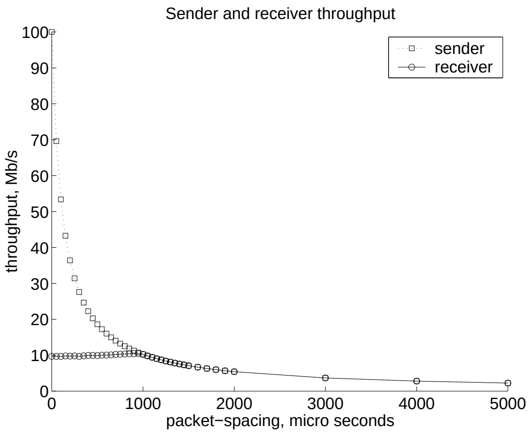

Figure 8. Throughput for One PSA of 16 PSAs and 16 TCP Connections

Figures 10 and 11 show the throughput and packet loss, respectively, of a representative TCP connection with its buffers tuned to the bandwidth-delay product. These figures show that with sufficient spacing by the PSAs, a TCP connection can consume its share of available bandwidth. For example, Figures 8 and 10 illustrate that with 3000 μs of packet spacing, each PSA receiver and TCP receiver achieves 5 Mb/s of throughput.

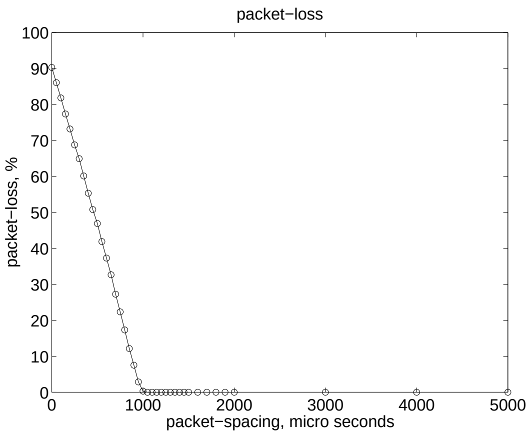

Figure 9. Packet Loss for One PSA of 16 PSAs and 16 TCP Connections

3.2.3. *PSA Spacing at the Receiver*

Assuming that applications create enough interleaving traffic to maintain packet spacing between communicating hosts rather than getting

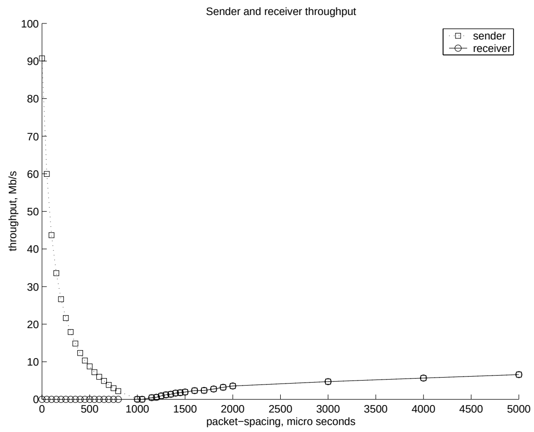

Figure 10. Throughput for One TCP of 16 TCP Connections and 16 PSAs

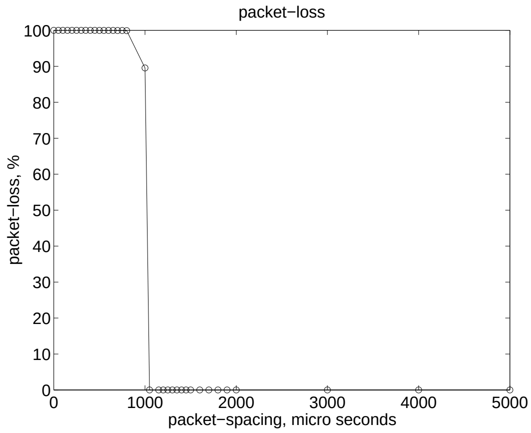

Figure 11. Packet Loss for One TCP of 16 TCP Connections and 16 PSAs

clumped as claimed by [13], we recorded the inter-arrival time of packets at one PSA receiver, using the same experimental set-up as described in Section 3.2.2. 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

First, we demonstrate how the damped PSAs try to find the ideal packet spacing under varying network conditions. Next, we show how the equation-based PSAs accomplish the same objectives as the damped PSAs, e.g., TCP-friendliness or fairness, while simultaneously providing a smoother sending rate than the damped PSAs. This smoother sending rate provides better support for time-sensitive applications such as video and audio streaming.

3.3.1. *Damped PSAs Competing*

When two damped PSAs compete for bandwidth, each damped PSA makes small oscillations around the ideal sending rate of 550 packets/RTT. Thus, each damped PSA gets its fair share of bandwidth. And even when the damped PSAs are started at different times, the sending rates quickly converge to 550 packets/RTT.

3.3.2. *Damped PSAs Competing with TCP*

In this simulation, we ran 10 TCP connections with infinite file transfers in the background and then added two damped PSAs to compete for bandwidth. Figure 12 shows that the damped PSAs respond readily to congestion. And again, both damped PSAs have very similar sending rates.

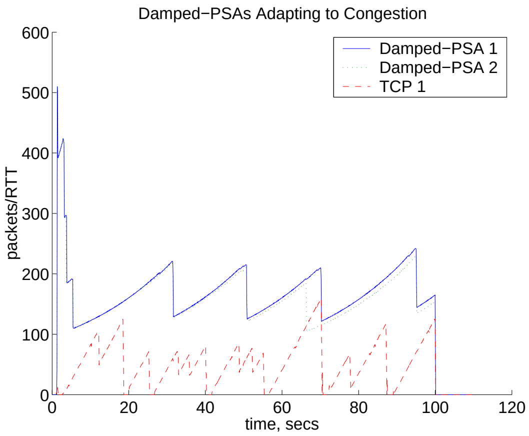

Figure 12. Two Damped PSAs Competing with Ten TCPs

However, the damped PSAs suffer from two problems. First, due to the "additive increase / multiplicative decrease" nature of our damped PSP, each damped PSA sees a significant rate change (i.e., halving of its sending rate) when a single congestion-indication event occurs. And unfortunately, for real-time applications such as video streaming, such a drastic change in sending rate is unnecessarily severe as it can noticeably reduce the user-perceived quality [25]. Second, while the damped PSAs are certainly "TCP-friendlier" that UDP packet blasting, the damped PSAs manage to claim approximately twice as much bandwidth as a representative TCP connection.

3.3.3. *Equation-Based PSAs Competing with TCP*

Similar to the previous section, we ran 10 TCP connections with infinite file transfers in the background and then added two equation-based PSAs to compete for bandwidth. In stark contrast to Figure 12, Figure 13 demonstrates that the equation-based PSAs produce a much smoother and fairer sending rate that is appropriate for multimedia applications. Furthermore, the packet-spaced sending rate of the equation-based PSAs is more TCP-friendly than the that of the damped PSAs or standard UDP packet blasting.

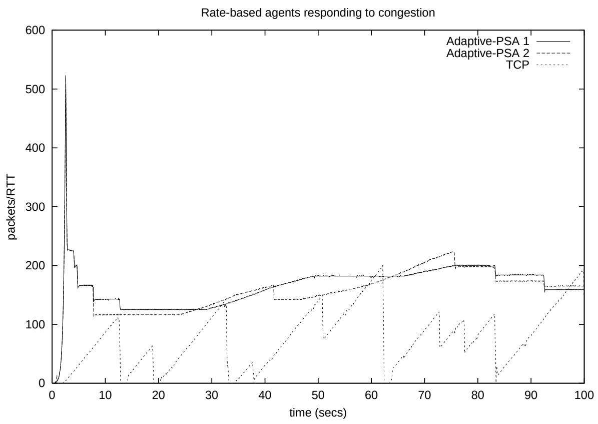

Figure 13. Two Equation-Based PSAs Competing with Ten TCPs

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 [7, 12]. 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 packet-spacing protocols are 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) 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. These initial results demonstrate that by introducing packet spacing to a multimedia stream, packet loss can be reduced dramatically while still maintaining decent (and relatively stable) throughput for multimedia applications.

To address the burstiness of the damped PSP and its topology-specific congestion-control algorithm, the equation-based PSP provides a more general mechanism for congestion control. The equation-based

PSP achieves fairness with TCP connections and results in very low packet loss and a smoother rate that is more appropriate for multimedia applications.

Future work includes examining the performance of the equation-based 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.

ACKNOWLEDGEMENT

This work was funded in part by the U.S. Department of Energy's Laboratory-Directed Research & Development Program and the U.S. Department of Energy's Office of Science.

References

1. 'UCB/LBNL/VINT network simulator - ns (version 2)'. http://www.isi.edu/nsnam/vint/index.html.
2. Aggarwal, A., S. Savage, and T. Anderson: 2000, 'Understanding the Performance of TCP Pacing'. In: *Proceedings of IEEE INFOCOM*.
3. Atwood, J. W. and G. C. K. Chung: 1993, 'Error Control in the Xpress Transfer Protocol'. In: *Proceedings of 18th Conference on Local Computer Networks*. pp. 423–431.
4. Baguette, Y. and A. Danthine: 1992, 'Comparison of TP4, TCP and XTP - Part 2: Data Transfer Mechanisms'. *ETT* **3**(5).
5. Childers, L., T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi: 2000, 'Access Grid: Immersive Group-to-Group Collaborative Visualization'. In: *Proceedings of the 4th International Immersive Projection Technology Workshop*.
6. Feng, A.: 2001, 'RAPID: Rate-Adjusting Protocol for Internet Delivery'. Thesis Proposal (in progress), University of Illinois at Urbana-Champaign.
7. Feng, W., D. Kandlur, D. Saha, and K. Shin: 2001, 'Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness'. In: *Proceedings of IEEE INFOCOM*.
8. Feng, W. and P. Tinnakornsrisuphap: 2000a, 'The Adverse Impact of the TCP Congestion-Control Mechanism in Heterogeneous Computing Systems'. In: *Proceedings of ICPP'00*.
9. Feng, W. and P. Tinnakornsrisuphap: 2000b, 'The Failure of TCP in Distributed Computational Grids'. In: *Proceedings of SC'00*.
10. Floyd, S. and K. Fall: 1999, 'Promoting the Use of End-to-End Congestion Control'. *IEEE/ACM Transactions on Networking*.
11. Floyd, S., M. Handley, J. Padhye, and J. Widmer: 2000, 'Equation-Based Congestion Control for Unicast Applications'. In: *Proc. of SIGCOMM 2000*.
12. Floyd, S. and V. Jacobson: 1993, 'Random Early Detection Gateways for Congestion Avoidance'. *IEEE/ACM Transactions on Networking* **1**(4), 397–413.

13. Jain, R.: 1992, 'Myths about Congestion Management in High Speed Networks'. *Internetworking: Research and Experience* **29**(2), 101–113.
14. Li, J., S. Ha, and V. Bharghavan: 1999, 'HPF: A Transport Protocol For Supporting Heregeneous Packet Flows in the Internet'.
15. Mahdavi, J. and S. Floyd: 1997, 'TCP-Friendly Unicast Rate-Based Flow Control'. Technical report, Note sent to end2end-interest mailing list.
16. Mo, J., J. Walrand, and V. Anantharam: 1999, 'Analysis and Comparison of TCP Reno and Vegas'. In: *Proceedings of IEEE INFOCOM*.
17. Padhye, J., V. Firoiu, D. Towsley, and J. Kurose: 1998, 'Modeling TCP Throughput: A Simple Model and its Empirical Validation'. In: *Proceedings of SIGCOMM'98*.
18. Padhye, J., J. Kurose, D. Towsley, and R. Koodli: 1999, 'A Model Based TCP-Friendly Rate Control Protocol'. In: *Proceedings of NOSSDAV'99*.
19. RealNetworks Inc.: 1999a, 'RBN^{TM} (Real Broadcast Network) White Paper'. Available at *http://www.real.com/solutions/-rbn/whitepaper.html*.
20. RealNetworks Inc.: 1999b, 'RealVideo Technical White Paper'. Available at *http://www.real.com/devzone/library/whitepapers/-overview.html*.
21. Rejaie, R., M. Handley, and D. Estrin: 1999, 'An End-to-End Rate-Based Congestion Control Mechanism for Real-Time Streams in the Internet'. In: *Proceedings of INFOCOM'99*.
22. Rhee, I., V. Ozdemir, and Y. Yi: 2000, 'TEAR: TCP Emulation at Receivers — Flow Control for Multimedia Streaming'. Technical report, North Carolina State University.
23. Sisalem, D. and H. Schulzrinne: 1998, 'The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme'. In: *Proceedings of NOSSDAV'98*.
24. Strayer, W. T., S. Gray, and R. E. C. Jr.: 1994, 'An Object-Oriented Implementation of the Xpress Transfer Protocol'. In: *Multimedia: Advanced Teleservices and High-Speed Communication Architectures, 2nd International Workshop (IWACA'94)*. pp. 387–400.
25. Tan, D. and A. Zakhor: 1999, 'Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol'. *IEEE Transactions on Multimedia*.
26. Tinnakornsrisuphap, P., W. Feng, and I. Philp: 2000, 'On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System'. In: *Proceedings of ICDCS'00*.