A priority-layered approach to transport for high bandwidth-delay product networks

Vidhyashankar Venkataraman[1] Paul Francis[1] Saikat Guha[1]

Murali. S. Kodialam[2] T. V. Lakshman[2] Manpreet Singh[1]

[1]Department Of Computer Science, Cornell University
[1]{vidya,francis,saikat,manpreet}@cs.cornell.edu

[2]Lucent Technologies
[2]{muralik,lakshman}@lucent.com

ABSTRACT

There have been many attempts at designing transport protocols that operate more aggressively than TCP for high bandwidth-delay product networks. Recent attempts exploit finer-grained feedback from the network, either explicitly as with XCP, or implicitly as with FastTCP. In this paper, we propose a new approach to increased aggressiveness that is complementary to previous approaches. We assign the packets from a given transport flow into two strictly prioritized flows. The higher priority flow operates with normal conservative AIMD. The lower priority flow, however, aggressively exploits spare capacity in the network while not interfering with AIMD flows. We provide simulation results that suggest that this priority-layered approach can yield high throughput even in a lossy network with limited buffers, and can considerably improve the completion time of short flows.

1 INTRODUCTION

There has been a sustained interest in the recent past to improve the performance of transport protocols over high bandwidth-delay product networks. These protocols operate by increasing congestion control aggressiveness by using MIMD instead of AIMD, while remaining stable and fair. FastTCP [3] does this by using MIMD only as long as delays do not increase, and moving slower when delays increase. XCP [2] does this by having routers monitor congestion and explicitly coordinate with sources to insure that in the aggregate, flows add the spare bandwidth available at the bottleneck.

This paper presents a new approach, called Priority-Layered Transport (PLT), to increasing aggressiveness. Specifically, PLT assigns packets from a given single transport flow into two strict priority classes. This results in two flows, a high-priority flow and a low-priority flow. The high-priority flow, which runs at the same priority level as competing TCP flows (i.e. best effort), operates conservatively (AIMD). The low-priority flow, however, operates aggressively, thus exploiting spare capacity in the network. Because the two flows are strictly prioritized, however, the aggressive low-priority flow never interferes with the conservative high-priority flow or legacy TCP flows. This isolation affords us considerable flexibility in the design of the aggressive low-priority component of the congestion control algorithm. For instance, the aggressive component itself need not necessarily be stable under all conditions, or even be fair with respect to the aggressive components of other transport flows. Put another way, this isolation insures that we won't do worse than regular TCP, and will often do substantially better.

The only requirements that PLT places on the network is that priority queuing, which is already implemented in routers as part of DiffServ, be 'turned on'. While this is not the case across the general Internet, it can easily be arranged for VPNs. The transport protocol at the end hosts must also be modified, but again this can be arranged in a private network.

This paper makes the following contributions.

- We propose a new approach to transport congestion control, PLT, that exploits priority queueing in the network to increase aggressiveness while remaining stable and fair.

- We describe a simple strawman protocol based on the PLT approach, and present preliminary simulation results which show that PLT considerably outperforms legacy TCP in throughput for long flows and completion time for short flows over lossy high

delay-bandwidth product networks [1].

The paper is organized as follows. Section 2 outlines some related work. Section 3 presents our strawman protocol design based on PLT, and section 4 presents the results of a simulated comparison of PLT and TCP. Sections 5 concludes the paper and shows directions for future work.

2 Related Work

TCP's limitations in high bandwidth-delay product networks were exposed in [4] and [5]; the authors had analytically proven that when the bandwidth-delay product of the network is high, it is difficult for TCP to maintain large average congestion windows even at low loss probabilities. Since then, there has been a host of alternative proposals to improve the performance of transport protocols in high bandwidth-delay product networks. Let us look at some of these protocols first in this section.

XCP ([2]) revamps the transport protocol to include support from the network in the form of an explicit feedback to the sender. This form of explicit congestion feedback is in direct contrast to the binary loss-based congestion signal used in TCP. An XCP-enabled router periodically calculates the spare capacity that should be utilized by the flows through it. The router then fairly distributes this capacity across all flows based on their respective congestion window sizes and RTT values and suggests the window change for each flow, as and when packets from the flows traverse through that router. The suggested value is overwritten on the packet only if the value from previous routers in the path is greater than the former. This ensures that the XCP sender eventually receives the window change from the bottleneck link and makes the necessary changes to its window size.

FastTCP ([3]) revived the interest in delay-based congestion protocols that had initially started with TCP-Vegas ([6]). Delay-based feedback offers much more control and stability to the window evolution than the oscillation-prone loss-based feedback. In FastTCP, the congestion window value for each flow is maintained such that it is just enough for the bottleneck link to get saturated and start queuing up. Though FastTCP cuts back aggressively during losses which are bound to happen when the network is subject to heavy dynamics, it ramps up multiplicatively, with the increase factor depending on the current congestion window size and the queuing delay; the factor reduces once queueing delay becomes non-zero. [3] shows that such a setup can lead to very fast convergence.

Apart from the two protocols mentioned above, there have been other protocols ([7], [8]) which augment the performance of TCP by altering its congestion algorithm to increase aggressively and decrease gracefully. But they are still unstable in the presence of high dynamics as shown in [3] and [9].

There have been some works in the past that make use of priority queuing in routers to enhance TCP's performance. TCP-Peach ([10]) is one such protocol designed for satellite networks. TCP-Peach uses 'dummy' segments at the low priority to probe the availability of bandwidth in bottleneck links. The sender gathers feedback from the receiver for these probes and increases its congestion window accordingly. PLT differs from TCP-Peach in actually sending data packets at the low priority so as to utilize the spare bandwidth at the bottleneck quickly.

Fast Start ([11]) modifies TCP's slow start to ensure faster completion of short and bursty data transfers. When a TCP connection is established, the initial values of TCP's parameters such as cwnd, ssthresh are set to the values based on the end host's cached history. If the initial value of cwnd is greater than unity, TCP enters the fast start phase wherein one packet is sent at a high priority while the remaining number (cwnd-1) of packets are sent through low priority. Fast Start lasts at most for a single RTT during the slow start phase and can stop prematurely if there are multiple losses. In contrast, the low priority traffic of PLT takes part during the entire course of a TCP connection so as to prevent the bottleneck from being under-utilized.

3 Protocol Description

We begin with an overview of PLT followed by a brief description of its components.

The schematic diagram of the protocol is shown in Figure 1. The sender side comprises of four basic modules. The two *control modules* (CM's) as shown in the figure, perform the central protocol processing functions namely flow control, congestion control, and error recovery. The two CM's are the high priority control module (HCM) which runs the legacy TCP congestion control

[1] A comparison study of PLT with other protocols specialized in high bandwidth-delay product networks is left as part of future work.

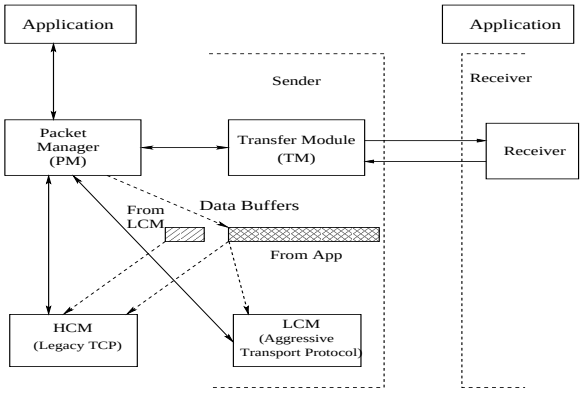

Figure 1: A schematic diagram of PLT.

algorithm and the low priority control module (LCM) which works in tandem with HCM. The packet manager (PM) assigns the next packet to send for each of the control modules. Finally, the transfer module (TM) sends and receives packets through the underlying network.

The receiver side is a TCP-SACK receiver with additional capabilities of marking ACK packets with priority levels and maintaining individual outstanding windows for the HCM and the LCM.

3.1 Congestion Control in LCM

LCM contributes to the aggressive behavior of PLT through its congestion control. LCM has to accomplish two goals in achieving this aggression: one is to sense if there is any under-utilized bandwidth in the bottleneck and the other is to quickly 'grab' this under-utilized bandwidth.

To achieve these goals, we have adopted a simple loss-rate-based congestion control algorithm at the LCM which is similar to the one proposed in [12]. According to the scheme, the LCM tries to limit its loss rate to at most a *target loss rate* μ, a threshold at which the LCM operates. The LCM changes its window size based on the loss rate that it has incurred in the recent past and the parameter μ. The LCM is aggressive as long as its observed loss rate is less than μ and cuts back when it incurs higher loss rates. Hence, in practice, the LCM should adaptively calculate μ based on the channel characteristics and the current loss scenario in the network. In this paper, however, we set μ statically.

Time is divided into epochs, each epoch roughly spanning the RTT of the flow (calculated using the existing approach in TCP). At the end of each epoch, the LCM calculates the loss rate (p) during that epoch as the ratio of the number of lost packets to the total number of packets for which the receiver had responded to. The overall loss rate (P) is then calculated as an exponential moving average. The LCM then reassigns its congestion window (cwnd) as:

$$cwnd = \begin{cases} \alpha.n + cwnd & \text{if P} < \mu \\ \beta.cwnd & \text{otherwise} \end{cases}$$

where α is the increase factor, β is the decrease factor, and n is the number of packets acknowledged by the receiver in the epoch. We see that the LCM uses an MIMD approach for congestion control; this is because an MIMD approach seizes an under-utilized channel faster than AIMD. Apart from this periodic window control, *cwnd* may have to be decreased when there is a timeout. So we introduce another parameter γ which is the decrease factor for the window cutback during timeouts.

3.2 The Transport Protocol

The packet manager (PM) receives data from the application and maintains the input buffers for the two control modules. The control modules send packets as long as their congestion window and the outstanding window sizes allow them to do so.

3.2.1 Error Recovery at the LCM

LCM adopts a simple error recovery mechanism. The LCM retransmits a lost packet some number of times after which it gets finally retransmitted at the HCM. In our current system, retransmissions do not occur at the LCM and we allow a lost LCM packet to be retransmitted directly at the HCM, for the sake of protocol simplicity. For this to happen, the LCM informs the loss to the PM which then simply enqueues the packet into a retransmit buffer maintained by the PM.

As we can see from Figure 1, the input buffer comprises of two sets of packets: the first is the queue of packets from the LCM waiting for retransmission while the second is the queue of non-transmitted packets from the application. The HCM prioritizes sending lost LCM packets over packets not yet transmitted. As for the LCM, it checks only the set of non-transmitted packets for the next packet to send[2].

[2]The LCM can make an exception by retransmitting a lost LCM packet if it finds the TCP window to be very small when compared to the size of the retransmit buffer.

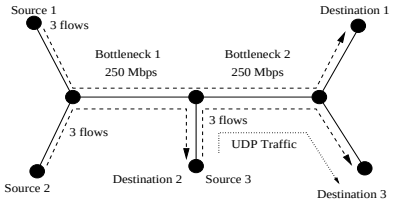

Figure 2: Multiple bottleneck topology.

3.2.2 ACK Mechanism at the receiver

The PLT receiver sends an ACK for every packet that it received. To ensure that the reverse traffic does not affect the other connections, the receiver can choose to send ACKs for the low-priority packets at the low priority. Such a design could potentially result in ACK losses, but will not adversely affect the performance of the protocol since TCP is known to be robust to ACK losses. In fact, to further reduce the low priority ACK overhead, the receiver can choose to use the delayed ACK mechanism (to send an ACK for every *n* low priority packets received).

4 PERFORMANCE

This section presents a preliminary ns-2 simulation study of PLT in high bandwidth environments. The HCM in these simulations is TCP-Reno.

The results of our simulations primarily highlight three important benefits of the protocol.
1) Sustained goodput with small buffers
2) Sustained goodput amidst losses
3) Fast completion of short flows

4.1 Experimental parameters

The following table lists the parameters invariant in our simulations, unless specified otherwise.

Parameter	Policy/ Value
Increase factor α	0.1
Decrease factor β	0.95
Timeout decrease factor γ	0.75
Bottleneck bandwidth (BBW)	250 Mbps
Max. Receiver Window	3.BW.RTT
Router Drop Policy	Tail drop
Epoch Duration	100ms
Simulation time limit	1200s
Target loss rate (μ)	0.5%

Most of the simulations presented in this paper are run on a single bottleneck link with a round-trip time of 80ms. To observe the effect of the protocol on the size of the bottleneck buffer, we experimented with both large and small buffer sizes: one equal to the bandwidth delay product (BBW.RTT≈2500) and the other, 10% of this product.

We have also conducted experiments on a topology with multiple bottleneck links (Figure 2). There are two bottleneck links each of which has a bandwidth of 250Mbps. There are nine competing TCP file transfers in the setup, arriving one after another, along with various rates of UDP traffic on the second bottleneck link to compete with the other flows.

4.2 Single flow

Figure 3 shows the goodput of a single flow with an infinite source and a single bottleneck link for various random loss rates for PLT and TCP. As expected ([4]), TCP fares very poorly even at a loss rate of 0.01%. By contrast, PLT yields very high goodputs even at a non-zero loss rates.

With zero loss rate, the window evolution of the HCM (TCP-Reno) shown in Figure 4 is well-known. At zero loss rates, the LCM window increases during those times when the HCM is not filling the bottleneck queue, but backs off otherwise.

As for the LCM window evolution at zero loss rate (Figure 5), it can expect to send packets successfully as long as the bottleneck is not saturated by HCM packets. But when the bottleneck queue starts getting filled up by the HCM packets, the LCM packets cannot make it to the receiver. Hence, the LCM times out and cuts back successively till either the LCM window size goes below one or the bottleneck gets under-utilized again (potentially due to a HCM cutback).

With non-zero loss rates (figures 4 and 5), the HCM never fills the queue, and so the LCM kicks in and utilizes the available capacity. This results in near-perfect goodputs as shown in Figure 3.

When we limit bottleneck buffer sizes (figures 6, 7, and 8), the role of LCM at zero loss rate is more pronounced, as the LCM exploits TCP's inability to keep the bottleneck buffer full at times. Figure 6 shows that PLT exhibits much higher goodputs than legacy-TCP.

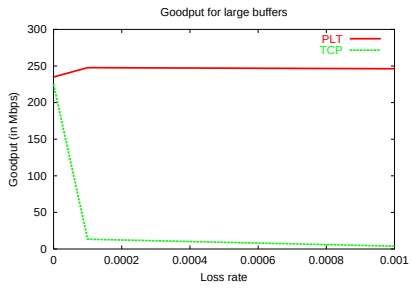

Figure 3: Goodput with large buffers

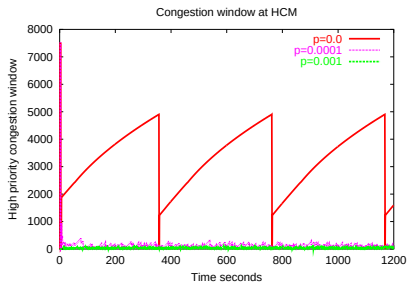

Figure 4: HCM window with large buffers

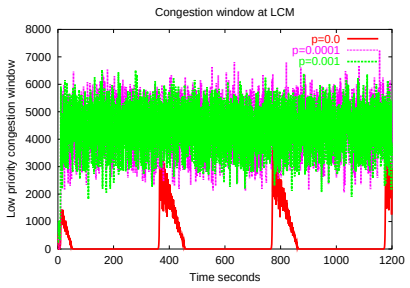

Figure 5: LCM window with large buffers

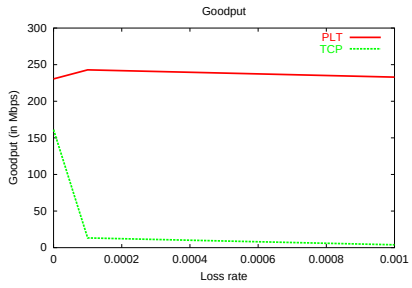

Figure 6: Goodput with small buffers

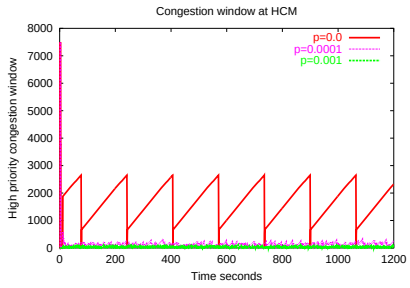

Figure 7: HCM window with small buffers

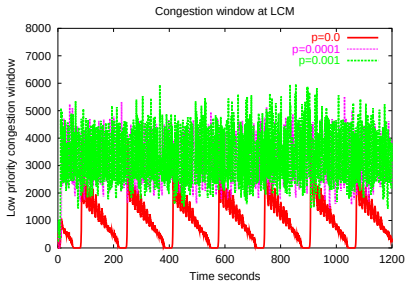

Figure 8: LCM with small buffers

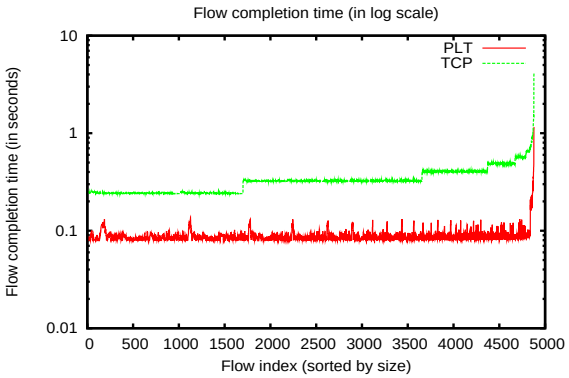

Figure 9: Short flow completion time

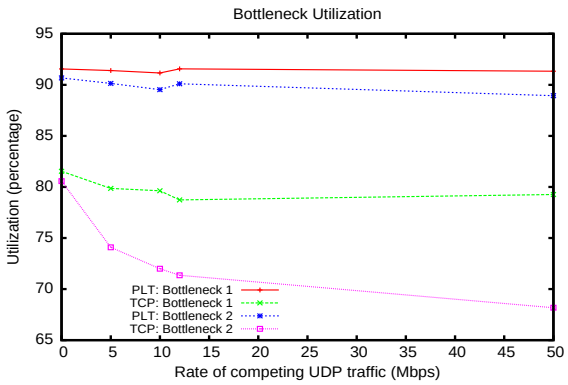

Figure 10: Efficiency over multiple bottleneck flows

4.3 Competing Flows

4.3.1 Completion time for short flows

We tested the completion time of flows arriving at a Poisson rate of 1250 flows per second: the rate is just enough to saturate the pipe. The volume of the flows is pareto distributed with an average size of 25 packets as is the case with Internet flows. Figure 9 shows the frequency distribution of the flow completion times, with the flows arranged in increasing order of size. We find that more than 90% of the PLT flows complete within a single RTT while TCP takes two or more RTTs to complete flows with similar sizes. While PLT's good performance here is of course due to the fact that the initial congestion window of the LCM is high enough to fill the pipe, we note that it is the fact that LCM packets run at lower priority that allows us to set the initial window size aggressively.

4.3.2 Multiple Bottlenecks

With the multiple bottleneck topology, we introduce UDP traffic to compete with the TCP flows hence resulting in cutbacks in TCP's congestion window and decrease in goodput. We observed the bottleneck utilization for various rates of the UDP traffic. Figure 10 shows that PLT can yield 90% average utilization in both the bottleneck links even at high UDP traffic rates. We find that with a 50Mbps UDP traffic, there is a 20% utilization increase with PLT, when compared to TCP. Since this utilization advantage is due to LCM and the retransmissions of LCM packets at the HCM account for less than

1% of the bottleneck bandwidth, we can conclude that PLT's aggregate goodput increases by 20% of the bottleneck bandwidth.

5 Conclusions and Future Work

This paper presents Priority-Layered Transport, a novel approach for increasing the aggressiveness of transport protocols while remaining stable and fair to TCP flows. This paper also presents preliminary simulations showing that a simple strawman version of PLT performs well better than legacy TCP. While we are encouraged by these results, it is hard to draw strong conclusions from this work other than to say the idea has promise. To draw broader conclusions, much work needs to be done.

First, we need to better understand the behavior of our own algorithms over a broad range of operating conditions. Specific questions include: Are PLT flows fair to each other, especially when losses force the aggressive low-priority component to dominate? Can the target loss-rate parameter μ be dynamically tuned? How should the initial window size be set so that it doesn't unduly interfere with existing PLT flows? One possible idea here is to set the initial window very aggressively (essentially try to saturate the link), but transmit these initial packets at a third, still lower priority level. Is legacy TCP a good choice for the conservative high-priority component, or would a more aggressive transport like FastTCP be a better choice?

Once we are convinced we have a good version of PLT, we need to analyze it. We also of course need to compare it head-to-head with the best-of-breed transports, including at least FastTCP and XCP. We also need to implement PLT in real systems and test it in emulated and real environments.

References

[1] S. Blake, D. Black, M. Carlson, E. Davie, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, RFC 2475, December 1998.

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

[3] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: Motivation, architecture, algorithms, performance. To appear in *IEEE/ACM Trans. On Networking*, 2007.

[4] T. V. Lakshman and U. Madhow. The performance of TCP/IP for neworks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. on Networking, 5(3):336-350*, June 1997.

[5] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Theoughput: A Simple Model and its Empirical Validation. In *The Proceedings of SIGCOMM*, 1998.

[6] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *The Proceedings of SIGCOMM* 1994.

[7] S. Floyd. Highspeed TCP for large congestion windows. RFC 3649, December 2003.

[8] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. In *The Proceedings of ACM SIGCOMM*, 2003.

[9] C. Jin, D. X. Wei, and S. H. Low. The case for delay-based congestion control. In *The Proceedings of IEEE Computer Communication Workshop (CCW)*, October 2003.

[10] I. F. Akyildiz, G. Morabito, and S. Palazzo. TCP-Peach: A New Flow Control Scheme For Satellite Networks. In *IEEE/ACM Trans. on Networking (TON)*, 2001.

[11] V. Padmanabhan and R. Katz. TCP Fast Start: A technique for speeding up web transfers. In *The Proceedings of GLOBECOMM 1998 Internet Mini-Conference*.

[12] J. Waldby, U. Madhow, and T. V. Lakshman. Total Acknowledgements: A Robust Feedback Mechanism for End-to-end Congestion Control. (Extended Abstract) *In SIGMETRICS 1998: 274:275*.