Packet Drop Avoidance for High-speed Network Transmission Protocol

Jin, Guojun

Distributed Systems Department
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley, CA 94720
g_jin@lbl.gov

Abstract: As network bandwidth continues to grow and longer paths are used to exchange large scientific data between storage systems and GRID computation, it has become increasingly obvious that there is a need to deploy a packet drop avoidance mechanism into network transmission protocols. Current end-to-end congestion avoidance mechanisms [1] used in Transmission Control Protocol (TCP) have worked well on low bandwidth delay product networks, but with newer high-bandwidth delay networks they have shown to be inefficient and prone to unstable. This is largely due to increased network bandwidth coupled with changes in internet traffic patterns. These changes come from a variety of new network applications that are being developed to take advantage of the increased network bandwidth. This paper will examine the end-to-end congestion avoidance mechanism and perform a step-by-step analysis of its theory. In addition we will propose an alternative approach developed as part of a new network transmission protocol. Our alternative protocol uses a packet drop avoidance (PDA) mechanism built on top of the maximum burst size (MBS) theory combined with a real-time available bandwidth algorithm.

I. Introduction

Basic TCP congestion control theory is well-known and a number of studies [2][3][4][6] to analyze it have been done in the past couple of years. Here, we take a different approach and analyze on the window-based congestion control mechanism of TCP.

Many people have worked on improving the TCP congestion control algorithm. TCP is unable to utilize all the available bandwidth on high-bandwidth and/or high-delay paths due to its conservative congestion avoidance algorithm. In fact TCP can become quite unstable under these conditions. One problem is the fact that TCP does not have a mechanism to distinguish between a slowest (narrow) link and congested (tight) link. This means that TCP's algorithm will continue to increase the congestion window (assuming tuned large buffers) to increase the sending rate as long as there is no further packet loss. This is problematic since packet drop could be caused by congestion at the narrow link. In either a high-speed and/or long delay path, when a congestion signal comes back to the sender, the outstanding data stream will be the average size of congestion window, which is computed from the acknowledgments during the last round-trip-time (RTT) period. Consider a 100ms RTT and 40Gb/s path, TCP needs to send a burst as large as 500 MBytes (333,333 1500Byte packets) of data during one RTT to detect congestion trend. This big burst of traffic plus existing cross traffic will exceed the bottleneck link router queue and cause up to 50% packet loss (more than 160K packets in above example). A self-clocking system could help reduce the loss probability when cross traffic is less bursty, but this may not be the condition under which the current network is dropping packets.

An examination of the congestion avoidance mechanism shows that we see bursts in two different phases of the TCP congestion control algorithm; slow start and congestion avoidance. In the slow start phase, the algorithm doubles the size of the burst until packet loss occurs, probing for the ceiling of the congestion window. After seeing packet loss, standard TCP congestion control reduces the congestion window to one half the current window size. If TCP sees more packet loss, it will reduce the window further. This is called "multiplicative decrease" which prevents further packets from causing collapse. This slow start algorithm assumes that a possible best congestion window is between the last burst (congestion window) and the previous burst (one half of the congestion window) since the previous burst did not cause packet loss. However, this does not efficiently avoid packet loss, especially when the bandwidth or path latency is high. For example, on a 100ms RTT and 100Gb/s path, the previous burst can be 1 GB, and doubling it can cause the increased 1GB data (666 thousand 1500B packets) loss. Since acknowledgments are asynchronously fed back to the sender, they can cause further fluctuations when the cross traffic is more dynamic. The key issue in the slow start phase is during the last few window adjustments. In a better TCP design, the last few probes should be used to

detect the bottleneck router's queue size and its capacity, and should not use an exponential increase of the burst (window) size to cause loss. Instead, it should use an adaptive algorithm to increase its burst size to avoid losing a large number of packets. This would also allow the detection of the best rate to pace out the packet. Window based congestion control mechanisms also lack the ability to predict congestion on-the-fly and dynamically adjust their sending rate to reflect the new available bandwidth.

In the rest of this paper, we address the theory of maximum burst size (MBS) and how to look at using it for improving the slow start phase and replacing the window-based congestion avoidance with burst-based packet loss avoidance. The new transmission protocol design also uses of FAC^2 [7], a real-time bandwidth availability algorithm to assist MBS for transmission pacing control. This algorithm can also be used to predict the bottleneck router's queue status, allowing one to design a better network transmission pacing control protocol.

II. Maximum Burst Size Theory

The maximum burst size (MBS) is a key mechanism to avoiding packet loss. At the application level, properly applying MBS can maximize the throughput of network applications. At the network engineering level, using MBS can help to avoid packet loss, which is the biggest enemy for network performance. The principle of maximum burst size theory is: "Any burst sent into the network that exceeds the MBS will potentially cause a router on the path to drop packets." The MBS value is determined from the stream sending speed, cross traffic rate and router queue size. The MBS is also called effective queue size. On an emulation network, where traffic can be controlled [http://dsd.lbl.gov/NCS/back/emn.html#EMN_LAB], MBS is analyzed and results are shown in figures 1-4.

Fig. 1 shows how MBS is measured and how a proper burst size smaller than MBS can avoid router queue overflow. The MBS measurement is illustrated by the curve (solid and smooth one) started from lower left corner at transmission start phase. Once the MBS probe process sees the potential queuing, it slows down the burst increase rate, where TCP slow-start keeps doubling its burst (congestion window) size. The typical MBS detection stops at the intersection of the curve and the double dash line. MBS then is updated during the entire transmission. In a situation when the available bandwidth of the high-speed path is close to DSL (digital subscribe line) or wireless bandwidth (also the delay is longer than normal), further probes may be needed to distinguish if the bottleneck is the slowest link (narrow link) or a congested link (tight link). An additional probe burst is sent in speed slightly higher than the current available bandwidth, and the burst size is slightly larger than current MBS. The narrow link will show more delay increasing than the tight link. If the bottleneck is the narrow link, transmission rate will be limited below the bottleneck bandwidth.

The polyline (top graph) in Fig. 1 shows router queuing caused by cross traffic. Horizontal rectangles (boxes) represent additional network traffic sent in either large bursts (a), medium bursts (b), or smaller bursts (c). aQ and bQ show that large bursts can easily cause queue overflow compared to medium bursts when suddenly increasing of cross traffic. Fig. 2 shows when encountering the same cross traffic, the larger burst has higher probability to cause packet drop.

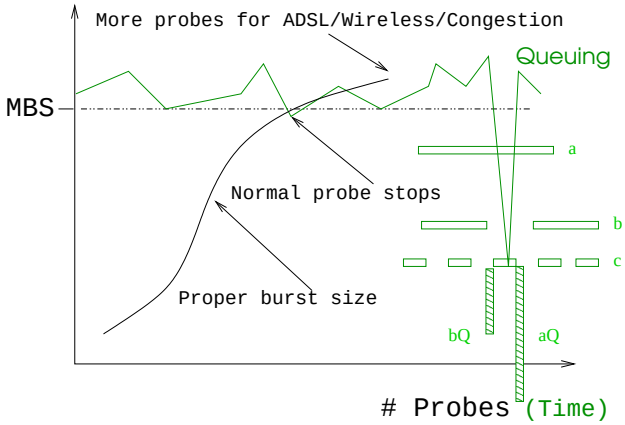

Fig. 1 Maximum Burst Size

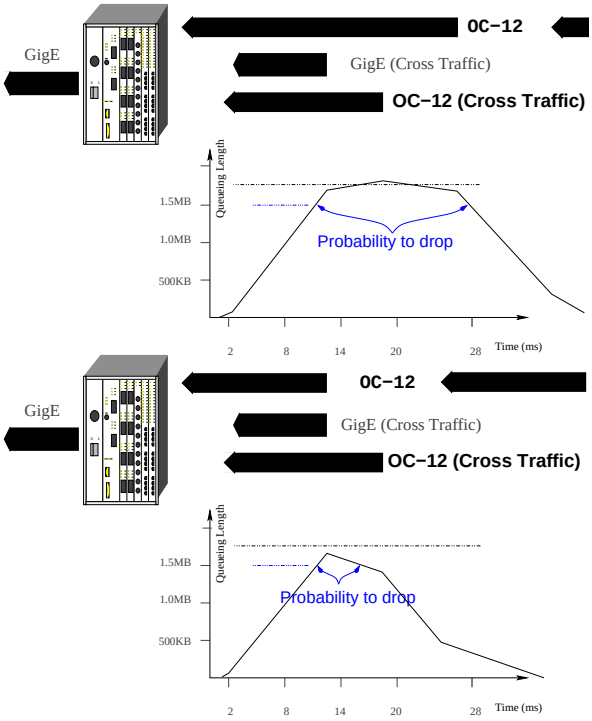

Fig. 2 smaller burst reduce packet drop probability

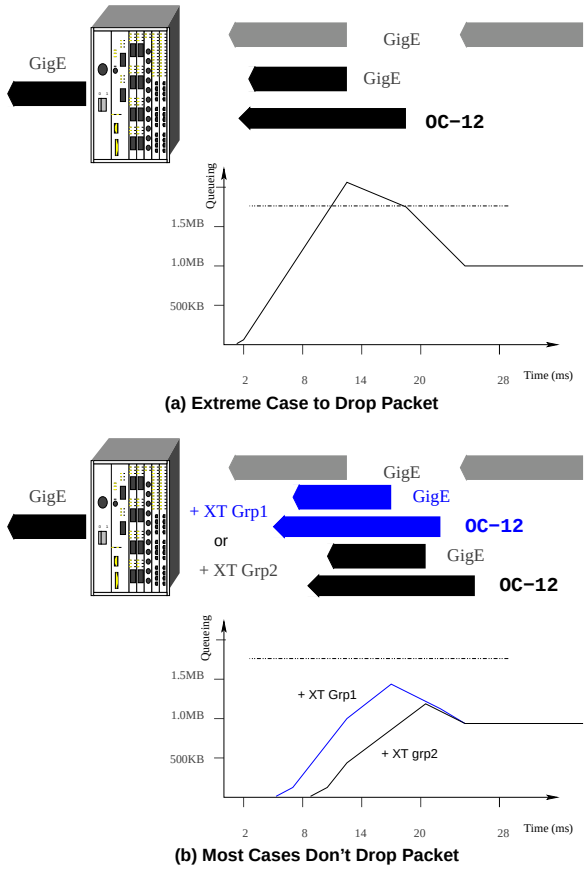

Fig. 3 MBS reduces packet drop chance

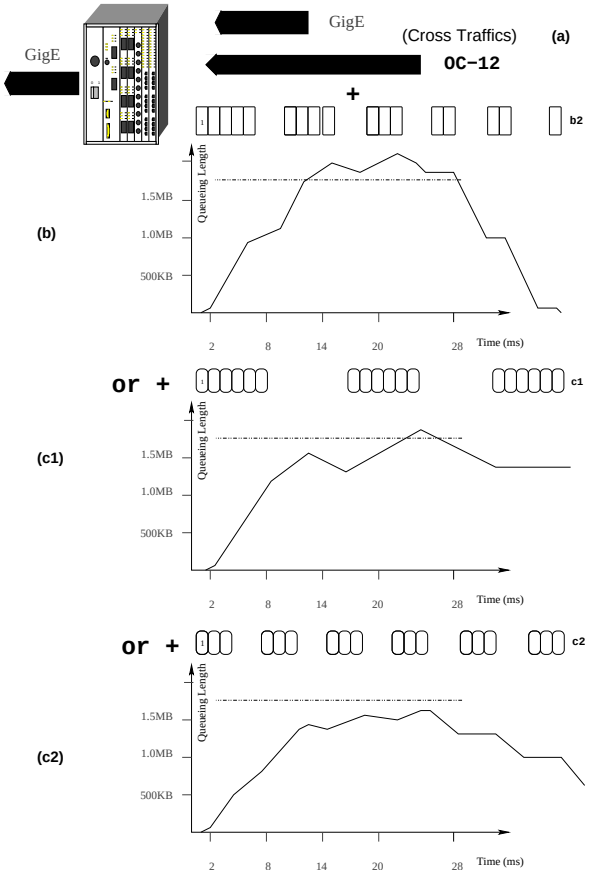

Fig. 4 Window vs. MBS pacing

Fig. 3 shows if packets are transmitted in small size bursts with proper gap, the chance to drop packets is narrowed only when all traffic arrives at a router at the same time. Fig. 3a shows when the first burst of a data stream (top of Fig. 3a) arrives to a 1Gb/s router with two cross traffic streams (1Gb/s in the middle, and Oc-12), these bursts can cause router queue overflow. In Fig. 3b, the same cross traffic streams are coming later (XT Grp1 started at 5ms), or coming even later (XT Grp2 started at 9ms), we can see the maximum queuing size is reduced. Fig. 2 and Fig. 3 illustrate that the larger the burst size is, the higher chance of packet loss will be. In the future, the bandwidth of a single network channel can be 1 Tb/s or higher. The following issues must be considered in network transmission protocol design for oversubscribing available bandwidth to saturate the physical bandwidth:

- A single host is not able to generate such high rate traffic.
- Saturation at a link is very unstable. That is, any additional traffic can cause packet loss.
- What percentage of network applications need full ultra high bandwidth?
- Can any algorithm stably keep a link 100% utilized?

If it is "True" for the first two issues, and the answer is "No" for last two questions, then, there is no need to design a transmission protocol to use a single stream to occupy entire network channel where bandwidth is ultra high (requiring bandwidth measurement algorithms to detect it). Furthermore, more and more applications will use Internet to exchange data and to share information, and many of them may generate constant traffic to the network. Avoiding packet loss is much more important than saturating a network channel. Therefore, adding a proper gap between bursts will not reduce a protocol efficiency, instead, it will minimize the packet drop probability, thus, increasing network transmission effectiveness. Fig. 3 has shown that choosing a proper burst size with a proper gap between bursts will effectively reduce packet loss, and this is a key issue in designing new network transmission protocols.

Fig. 4 shows that packet drop probability in transferring similar amount of data between MBS-based pacing control vs. Window-based pacing control. The top graph (a) shows two background (cross) traffic streams. Fig. 4b shows window-based transmission control; the middle graph shows larger burst-based transmission control, and

the bottom graph is the smaller burst-based transmission control. In window-based transmission control, the middle 8 packets (start from packet 7) are experiencing queue overflow. The larger burst-based transmission control (Fig. 4c1) experiences one potential loss or causes other traffic to drop a packet, where the smaller burst-based transmission control (Fig. 4c2) sends 18 packets without causing any packet loss in all traffic. The entire Fig. 4 demonstrates that dynamically changing burst size has less efficiency than using proper burst size with proper gap to reduce packet loss. In Fig. 4b, we assumed that the congestion information was timely fed back to the sending host to adjust the window (burst) size, it still experienced 20%~25% packet loss (loss is distributed in two streams). In fact, the congestion information is fed back after one round trip time (RTT). In high bandwidth delay product path, this will cause massive packet loss (example in §I.). With retransmission and congestion recovery, window-based transmission protocol not only has poor throughput, but also reduces available network bandwidth to other traffic. MBS-based transmission pacing can effectively avoid packet loss by choosing proper maximum burst size, and it controls transmission pace by adjusting gaps between bursts. Avoiding collision on highly utilized highway is analogy to this idea. The higher the traffic speed is, the larger gap between cars is needed.

III. Alternative Approach for Network Transmission Protocol

XCP (eXplicit Control Protocol)[3] is meant to provide a mechanism for revising current network transmission control protocols. XCP does this by providing explicit information from a router, thus allowing TCP (or any protocol) a greater degree of congestion control. In our protocol we use FAC^2 [7] to actively gain the same information provided by XCP and to also deduce additional information (e.g. bottleneck router status). This is done by measuring current traffic flows without injecting additional traffic (which can cause perturbations to the network). The transmission packets are then paced out by proper MBS. It should be noted that active queue management (AQM) does not affect MBS-based transmission control protocols because the protocol knows the current available bandwidth, understands the cause of packet loss, and does not react to random loss to do multiplicative decrease.

An alternative network transmission protocol we named Network Lion [8] (a name picked for a strong and robust network protocol). Lion does not just redesign the transmission control, but also separates the pacing control in layer 3 and retransmission control in layer 4. The two main reasons for moving transmission pacing to layer 3 are able to control all upper layer traffic (not just reliable stream) flowing to the same bottleneck router, and to prepare for system on chip (SoC). SoC is done to reduce the burden on CPU and memory bandwidth, and obtain better traffic pacing control. It is difficult to move retransmission control onto a chip because on a high bandwidth delay product path, unacknowledged data can be more than 1GBytes (for example, a path with100 Gb/s bandwidth and 80 ms RTT), and building such large static memory onto a chip seems to be fantasy. This means that putting entire TCP/IP on chip is not a proper design, thus, transmission pacing control should go into layer 3 (prepare for SoC) and retransmission (reliable/non-reliable) control should be stay at layer 4.

IV. Experiment

This new transmission protocol has been implemented in the FreeBSD kernel for experiment. Network Lion implements all of the above improvements while still maintaining the ability to compatible with all current TCP hosts. In this section we will show results of the experimental implementation of Lion, comparing it to the performance of standard TCP in Linux and FreeBSD. Fig. 5 shows the transmission sequence and performance of a Linux 2.4.19 TCP implementation (top graph) and our Lion implementation under FreeBSD 4.8 (bottom graph). Both sending hosts are 2.0GHz Intel P4 Xeon. The test was performed by starting two *iperf* clients on hosts at LBNL and directed toward the same destination at PSC (an 80ms RTT path) over a GigE WAN. Three different phases show up in Fig. 5, slow start, detecting other traffic and drop avoidance. The Lion (lower graph) has a very short start phase, where as the Linux TCP has a standard slow start. At phase two (time index 16:06.47.0), Lion detects the lower available bandwidth and starts to yield a percentage of the used bandwidth to other traffic. This is a shuffle algorithm in Lion that allows it to redistribute the bandwidth. The algorithm assumes all existing traffic is Lion, and after detecting new traffic, yields a scaled bandwidth based on utilized bandwidth from existing streams. For example, throughput between 500Mb/s~999Mb/s yields 200 Mb/s, throughput between 150Mb/s~499Mb/s yields 100 Mb/s, and so on. This allows each stream to obtain the number of existing streams by evaluating the yielded bandwidth, then each stream can quickly adjust to the new pacing rate for itself. In this test, Linux TCP sped up after seeing more bandwidth yielded from Lion. This sent a signal to Lion that there is no way to detect the number of streams and the competing traffic is too aggressive. This caused Lion to

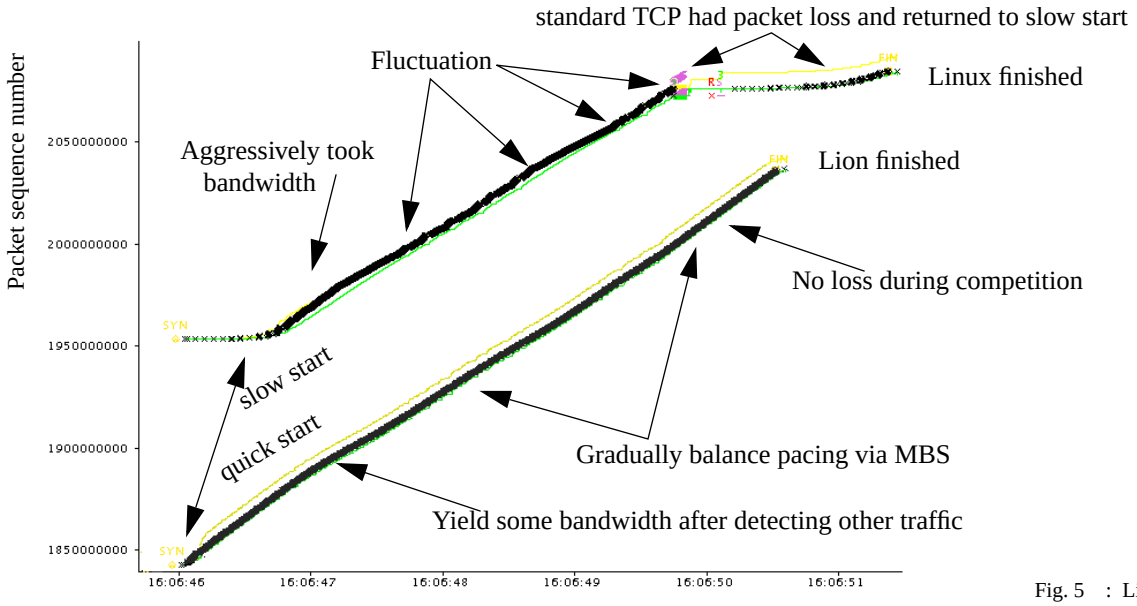

Fig. 5 : Lion vs. Linux TCP

balance the bandwidth by gradually increasing the burst pacing. From 16:06:47.75, we see Linux TCP sending rate going down and Lion sending rate ramping up. At 16:06:49.6, the Linux TCP experienced packet loss, which sent Linux TCP back to slow start, whereas Lion quickly finished the data transfer. After Lion finished, the Linux TCP is still slowing, increasing its congestion window. The Lion gained about 327 Mb/s throughput while in contrast the Linux TCP only obtained about 175 Mb/s throughput during this 5-second contest. Since this test is relatively short, slow start is one reason Linux TCP performance never caught up. Yet, if we look at the time from 16:06:49.6, we can see that the main factor that affected the performance is packet loss and re-adjustment of the transmission pace. In contrast, Lion can detect the available bandwidth, so losing a packet will not affect its transmission pacing.

V. Summary

As networks evolve, reducing the packet loss rate is going to be a priority issue if we want to increase stability of network transmission protocols. The maximum burst size (MBS) provides a theory to reduce the probability of dropping packets for each stream and to quickly react to abrupt traffic changes. We have shown that building a reliable network transmission protocol needs algorithms like FAC^2 to measure the current transmission traffic to obtain the available bandwidth. By applying this knowledge with experiments that use MBS-based packet drop avoidance (PDA) to replace window-based congestion avoidance we have shown how to improve the efficiency and stability of network transmission protocols.

VI. Acknowledgments

This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division under U.S. Department of Energy Contract No. DE-AC03-76SF00098. This is report no. LBNL-53920

References:

[1] Jacobson, V. Congestion avoidance and control. In Proceedings of SIGCOMM '88, Stanford, CA, Aug. 1988

[2] Sally Floyd, HighSpeed TCP for Large Congestion Windows and Quick-Start for TCP and IP, Yokohama IETF, July 18, 2002, Available at http://www.icir.org/floyd/hstcp.html

[3] Dina Katabi, Mark Handley, Charlie Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. SIGCOMM ‘02, Pittsburgh, Pa, Aug. 2002

[4] Tom Kelly, Scalable TCP: Improving Performance in Highspeed Wide Area Networks Submitted for publication, December 2002.

[5] G. Jin, B. Tierney Netest: A Tool to Measure Maximum Burst Size, Available Bandwidth and Achievable Throughput, Proceedings of the 2003 ITRE, Newark, NJ, Aug. 10-13, 2003, LBNL-48350.

[6] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, FAST TCP: From Theory to Experiments, Dec. 6, 2003, available at http://netlab.caltech.edu/FAST/publications.html

[7] G. Jin, Feedback adaptive control and feedback asymptotic convergence algorithms for measuring network bandwidth. LBNL-53165

[8] http://dsd.lbl.gov/~jin/network/lion