Published in the Proc. of the 14th Annual ACM/IEEE SC2001 Conference, November 2001. LA-UR 01-2823

Dynamic Right-Sizing: TCP Flow-Control Adaptation

Mike Fisk[§†]
mfisk@lanl.gov

Wu-chun Feng[§‡]
feng@lanl.gov

[§]Los Alamos National Laboratory[*]
[†]Department of Computer Science & Engineering, University of California, San Diego
[‡]Department of Computer & Information Science, The Ohio State University

1 Introduction

Grid and networking researchers continue the practice of manually optimizing TCP buffer sizes to keep the network pipe full [10, 7], and thus achieve acceptable performance over the wide-area network, whether for bulk-data transfer or in support of computational grids [4], data grids [1, 8, 3], or access grids [2]. Not only is this process cumbersome, but the result of tuning window sizes for a particular pair of hosts is sub-par performance for connections with larger delay-bandwidth products and the misappropriation of scarse resources to connections with smaller delay-bandwidth products [9].

Consequently, we propose an operating system technique called Dynamic Right-Sizing that eliminates the need for this manual process. Compared to previous work on this problem, our solution is more efficient and both more transparent and widely-usable to applications.

The Web100 project has released a modified FTP client that uses user-space code to send a burst of pings to estimate the latency and bandwidth at the beginning of a connection and adjust the windows accordingly [6]. This approach requires that this measurement code be deployed in each application. Further, it uses a measurement period before data is sent and creates extra network traffic that is not controlled by a congestion avoidance mechanism. Thus, it can only be used sparingly, such as at the beginning of a connection. In contrast, our implementation automatically benefits not just FTP, but every application on a host.

Earlier work in [9] presents kernel modifications for 'auto-tuning' a sender's flow-control window based on the congestion window and then using fair-share algorithms to manage competition between connections for buffers. In this scheme, the receiver's window advertisements are superfluous. This solution does not adequaquately solve the problem for applications such as archival storage where the bottleneck in a connection is the receiving application instead of the network. In this context, the receiver still needs flow control or else unbounded amounts of receiver buffer space can be consumed by data that has been received and acknowledged by the operating system, but is waiting for the receiving application. This could in turn require the receiver to resort to dropping packets and unnecessary triggering of the sender's congestion avoidance mechanisms, thus wasting bandwidth and reducing throughput. Our technique complements the 'auto-tuning' presented in [9] by providing the receiver with the ability to measure the size of the sender's congestion window, and more fairly allocate buffers to connections based on their need for buffers.

2 Dynamic Right-sizing

In short, *Dynamic Right-sizing* lets the receiver estimate the sender's congestion window size and use that estimate to dynamically change the size of the receiver's window advertisements. As a result, the sender will be congestion-window-limited rather than flow-control-window-limited.

A sender can send no more than one window's worth of data between acknowledgements. Accordingly, a burst that is shorter than a round-trip time can contain at most one window's worth of data. Thus, for any period of time that is shorter than a round-trip time, the amount of data seen by the receiver over that period is a lower-bound on the size of the sender's window. Some data may be lost or delayed by the network, so the sender may have sent more than the amount of data seen. Further, the sender may not have had a full window's worth of data to send. So the window may be significantly larger than this lower-bound, but not if the connection is truly limited by the receiver's window. Measuring this minimum and making sure that the receiver's advertised window is always larger will let the receiver track the congestion window size.

To make these measurements, it is necessary for the receiver to know the round-trip time. In a typical TCP implementation, the round-trip time is measured by observing the time between when data is sent and an acknowledgement is returned [5]. But during a bulk-data transfer, the receiver might not be sending any data and would therefore not have a good round-trip time estimate. For instance, an FTP data connection transmits data entirely in one direction. A system that is only transmitting acknowledgements can still estimate the round-trip time by observing the time between when a byte is first acknowledged and the receipt of data that is at least one window beyond the sequence number that was acknowledged. If the sender is being throttled by the network, this estimate will be valid. However, if the sending application did not have any data to send, the measured time could be much larger than the actual round-trip time. Thus this measurement acts only as an upper-bound on the round-trip time and should be be used only when it is the only source of round-trip time information.

[*]This work was supported by the U.S. Dept. of Energy's Next Generation Internet - Earth Systems Grid and Accelerated Strategic Computing Initiative - Distance and Distributed Computing and Communication programs through Los Alamos National Laboratory contract W-7405-ENG-36.

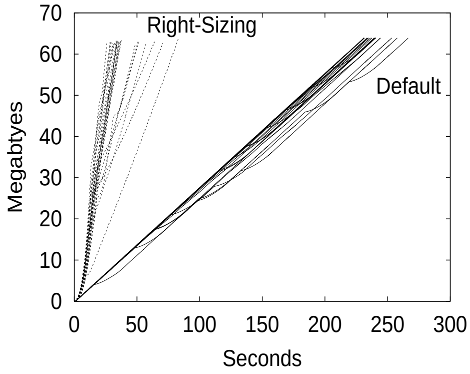

Figure 1: Progress of data transfers

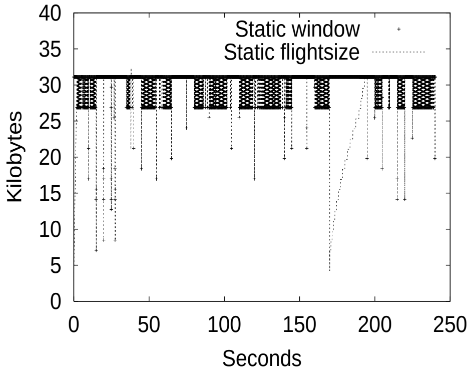

Figure 2: Default window size
Flight & window sizes

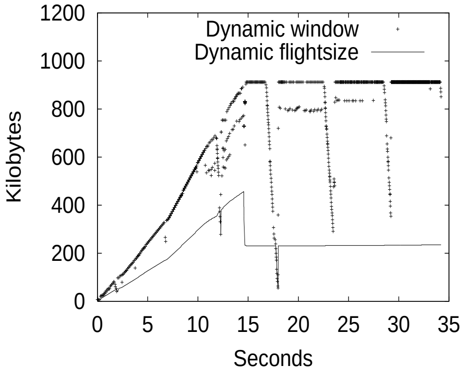

Figure 3: Dynamic right-sizing
Flight & window sizes

2.1 Impact of Timer Imprecision

Like many systems, the system timer in Linux is limited to a precision of 10ms. In this section we explore the impact of using this efficient, but imprecise measure of time.

Assume the salient scenario where a sender is always sending as fast as possible, but that the network may delay packets arbitrarily. However, we are concerned with the case were the sender is limited by the window-size, so that network delays are small enough that no timeouts occur. Further assume that the window-size is not fluctuating during a measurement. Then the receiver can determine that the window size limiting the sender is bounded as follows:

$$\frac{d}{n_{max}} \leq w \leq \frac{d}{n_{min}} \tag{1}$$

Where d bytes of data have been received over some number of round-trip times between n_{min} and n_{max}.

Any measurement period consists of some number of whole round-trip times plus fractional round-trip times preceding and following the complete round-trips. The possible number of round-trip periods observed is a whole number bounded as follows:

$$\left\lceil \frac{t}{rtt} \right\rceil \leq n < \left\lfloor \frac{t}{rtt} \right\rfloor + 2 \tag{2}$$

Due to similar fence-post problems, a measurement of duration equal to one round-trip time may actually be up to 20ms longer than the round-trip time. Combining these facts with bounds 1 and 2 yields:

$$\frac{d}{\left\lfloor \frac{20\text{ms}}{rtt} \right\rfloor + 3} \leq w \leq d \tag{3}$$

Thus, in no case will the actual window be larger than the measured amount of data received during the period. However, the amount of data received during the period may be three times the actual window size when measurements are made across wide-area networks with $rtt > 20$ms. Further, local networks with small round-trip delays may be grossly over-estimated.

We therefore conclude that measurements made with coarse timers will not cause dynamic right-sizing to underestimate the window size or negatively impact throughput. However, to make more accurate decisions for memory sharing under pressure, it is advantageous to use precise timers. Many CPU architectures now feature hardware time counters that can be used to efficiently obtain a precise timestamp. In future versions of our implementation, we will pursue the use of these counters. However, our use of these counters will also require that the standard TCP round-trip time estimation be done with equivalent precision.

3 Experimental Performance

In Figure 1, 50 transfers were made between Linux 2.2 systems modified to perform Dynamic Right-Sizing and connected by a network with an emulated round-trip delay of 100ms. The first 25 transfers used the default window sizes of 64 kilobytes for both the sender and receiver. The second 25 transfers, shown in dotted lines, used the dynamically sized windows described above.

In Figures 2 and 3, we examine the window sizes during two of the above transfers. The amount of sent, but unacknowledged data in the sender's buffer is known as the flightsize. The flightsize is in turn bounded by the window advertised by the receiver. Figure 2 shows that in the traditional, static case without dynamic right-sizing, the congestion window, and consequently flightsize quickly grow equal to the size of the window advertisements. For most of the duration of the connection, it is limited by the receiver's low window advertisement of 32KB.

In contrast, during the dynamic right-sizing case shown in Figure 3, the receiver is able, during most of the connection, to advertise a window size that is roughly twice the largest flightsize seen to date. As a result, the flightsize is only constrained by the congestion window and the delay-bandwidth product. Slow start continues for much longer and stops only when there is packet loss. At this point the congestion window stabilizes on a flightsize that is 7 times higher than the constrained flightsize of the static case. This 7-fold increase in the average flightsize is the source of the same, 7-fold increase in throughput demonstrated in Figure 1.

References

[1] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke, "The data grid: Towards an

architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, July 2000.

[2] Lisa Childers, Terry Disz, Robert Olson, Michael E. Papka, Rick Stevens, and Tushar Udeshi, "Access grid: Immersive group-to-group collaborative visualization," in *Proceedings of the 4th International Immersive Projection Technology Workshop*, 2000.

[3] "EU-data grid," http://www.eu-datagrid.org/.

[4] Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1998.

[5] Van Jacobson, "Congestion Avoidance and Control," in *Proceedings, SIGCOMM '88 Workshop*. ACM SIGCOMM, Aug. 1988, pp. 314–329, ACM Press, Stanford, CA.

[6] Jian Lui and Jim Ferguson, "Automatic TCP socket buffer tuning," in *Supercomputing 2000 Research Gems*, Nov. 2000, Awarded "Best Research Gem of the Conference," http://dast.nlanr.net/Features/Autobuf/.

[7] Jamshid Mahdavi, "Enabling high performance data transfers on hosts," Webpage, http://www.psc.edu/networking/perf_tune.html.

[8] "Particle physics data grid," http://www.cacr.caltech.edu/ppdg/.

[9] Jeff Semke, Jamshid Mahdavi, and Matt Mathis., "Automatic TCP buffer tuning," *Computer Communications Review*, vol. 28, no. 4, pp. 315–323, Oct. 1998.

[10] Brian L. Tierney, "TCP tuning guide for distributed application on wide area networks," *;login*, vol. 26, no. 1, Feb. 2001.