Published only as a Los Alamos Unclassified Report LA-UR 00-3321

Dynamic Adjustment of TCP Window Sizes

Mike Fisk[†]
mfisk@lanl.gov

Wu-chun Feng[†§]
feng@lanl.gov

[†]Los Alamos National Laboratory *
Los Alamos, NM 87544

[§]Purdue University
West Lafayette, IN 47907

Abstract

The original design of TCP failed to support reasonable performance over networks with large bandwidths and high round-trip times. Subsequent work on TCP has enabled the use of larger flow-control windows, yet the use of these options is still relatively rare, because manual tuning has been required. Other work has developed means for avoiding this manual tuning step, but those solutions lack generality and exhibit unfair characteristics. This paper introduces a new technique for TCP implementations to dynamically and automatically determine the best window size for optimum network performance. This technique results in greatly improved performance, a decrease in packet loss under bottleneck conditions, and greater control of buffer utilization by the end hosts. Previous research on buffer management can then be applied to these buffer allocation decisions.

1 Introduction

Over the past decade or more, TCP has entrenched itself as the ubiquitous transport protocol for the Internet. Virtually all TCP variants (e.g., Tahoe, Reno, Vegas) distinguish themselves on the basis of their congestion-control mechanisms. For much of this history, very little work has been done to the TCP flow control mechanisms.

Rapidly increasing bandwidths and delays have only in the last few years created a general need for increased attention to flow-control mechanisms. High-speed wireless, satellite, and long-haul terrestrial connections are now common. To fully utilize these networks, flow-control mechanisms must support the large delay-bandwidth products of these networks.

The ability to use larger flow-control windows has been provided with the addition of window scaling [1] and selective acknowledgements [2]. More recent work has included the problems of actually managing the windows and buffers on end-systems.

To this day, the size of most flow-control windows is set to operating system defaults. The state of the art for tuning these windows has depended on application- or user-specified buffer sizing using tools such as Pathchar [3] for characterizing network links by making measurements that are both time- and bandwidth-consuming.

Later work has introduced the notion of including dynamic tuning in the TCP implementation itself [4]. This work has focused on the common case in which the receiver window is largely unused and can consequently be ignored. In its place, optimistic fairness algorithms are used to maximize the size of the windows on the sender. This tuning performs well in many cases, but, as shown later, can induce unfair packet loss.

To date, there has yet to be a study of a generic algorithm for maintaining end-to-end flow control while automatically providing windows that scale to meet the link characteristics. Such a technique is described below. Test results demonstrate that equivalent performance gains can be achieved without sacrificing the ability to perform guaranteed, end-to-end flow control.

*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. This paper is Los Alamos Unclassified Report (LAUR) 00-3321.

2 Sliding Window Flow Control

Sliding window protocols such as TCP use a sliding window to avoid stop-and-go behavior by pipelining the delivery of data. The basic rules of engagement are that the receiver advertises a window size and the sender can send up to that amount of data. No more data can be sent until the first part of the data is acknowledged.

2.1 Window stalling

In every packet, a TCP end-point advertises the amount of window space currently available. If that window size, minus any unacknowledged data that has already been sent to that system, is zero, then no more data can be sent until a larger window is advertised.[1]

Thus, a full window will cause the sender to stall even if it has data to send and its congestion window would permit it. For the sender to be able to send data constantly, the receiver's window must always exceed the amount of sent, but unacknowledged data.

Figures 1 and 2 demonstrate the effect of changing the window size. In the first figure, the window is half the size necessary to keep the network fully utilized. As a result, the transmission stalls and is bursty.

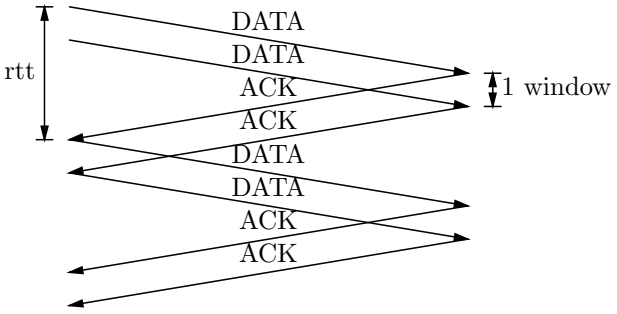

Figure 1: Stop-and-go

In the second figure, the window is the correct size. The result is that in the same period of time twice as much data has been received.

[1] One-byte packets can be sent to trigger a response with the current window size.

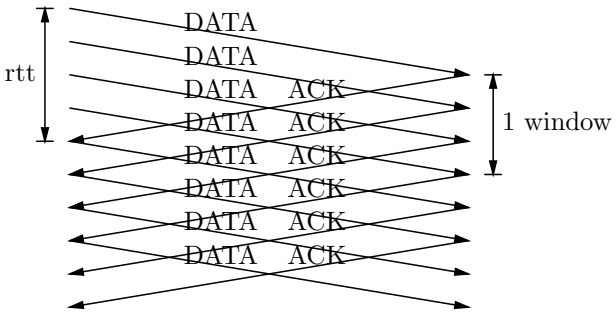

Figure 2: Full window

2.2 Theoretical sizing

Assuming that the receiver is able to immediately acknowledge transmitted data, the maximum amount of data that can be in transit is limited to the delay-bandwidth product of the network. In practice, packets may be delayed in queues for variable amounts of time. Further delay that may be introduced by delayed acknowledgements and other TCP implementation details. Fortunately, common techniques for measuring round-trip times include these delays.

In practice, the window size is usually representative of the amount of free buffer space that the receiver has committed to this connection. Assuming that the receiver is able to deliver acknowledged data to the application at the same rate at which is received, the receiver should never have a extensive amount of data buffered. As a result, the advertised window size will approach the size of the receive buffer.

3 Uniform Buffer Sizes

Since the inception of TCP, the bandwidth-delay product of most wide-area networks has been relatively small. The window (and consequently buffer) size necessary to allow full bandwidth utilization has been small enough that hosts have allocated fixed-sized send and receive buffers to each connection. Over time, operating systems have changed the default buffer size from common values of 8 kilobytes to as much as 64 kilobytes. The growth in host memory size has more than kept pace with the growth in bandwidth.

3.1 New trends

The growing use of the Internet and distributed applications is resulting in faster growth in available and desired bandwidth. OC-3 (155 megabits/second) and even OC-12 (622 megabits/second) connections are becoming commonplace. Meanwhile the number of users with 56 kilobit or less connections also continues to grow.

Current performance optimizations for web servers and operating systems are targeting the support of more than 10,000 simultaneous connections to a server. [5]

Meanwhile, the speed of light remains the same and round-trip delays of satellite links still reach 500ms. Cross-continent terrestrial links are frequently on the order of 100ms.

3.2 Cost of uniform buffer sizes

Consider the case of a heavily-used server on the Internet. This server may quite reasonably have a 622 Mb/s OC-12 connection to the Internet. Many of its clients may be across a continent or even around the world. For the sake of argument, we consider common, 100-millisecond, intra-continental delays in the discussion below.

If maximum performance is to be achieved with a client on a 100 Mb/s network connection, a window of 1.25 megabytes (0.1s × 100 Mb/s) is necessary. Meanwhile, the majority of the server's connections may be to clients with 56 kilobit network connections. These clients can only make use of a 0.7 kilobyte (0.1s × 56 kb/s) window. Table 1 shows the window sizes for these and several other types of links. Figure 3 graphically depicts the magnitude of the difference between these links.

The current TCP standard [6] requires that the receiver must be capable at all times of accepting a full window's worth of data. If the receiver over-subscribes its buffer space, it may have to drop an incoming packet. The sender will discover this packet loss and invoke TCP congestion control mechanisms even though the network is not congested. It is clear that receivers should not over-subscribe buffer space if they wish to maintain high performance.

If operating systems were to uniformly use a buffer size that could fully serve the above client case, each connection would be allocated a 1.25 megabyte buffer. To support large transfers in both directions, two such buffers would be required for each connection. With 10,000 simultaneous connections, the server would need over 24 gigabytes (10,000 × 1.25 MB × 2) of memory just for the socket buffers. This is not only costly, but larger than the addressable memory of a typical 32-bit server.

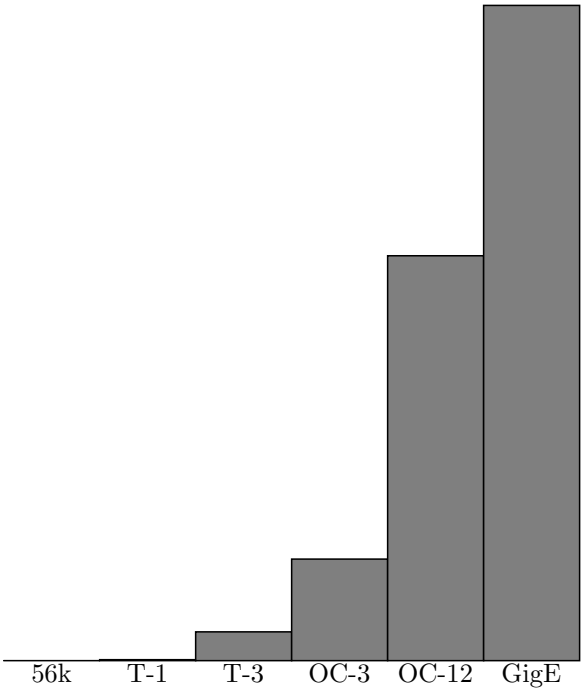

Figure 3: Relative Delay-Bandwidth Products

If a currently common window size of 64 kilobytes were used, 1.2 gigabytes of memory would still be required. If, instead, the 0.7 kilobyte buffers necessary for dialup clients were uniformly used, a trivial 14 megabytes of memory would be needed.

It is clear that when using uniform window sizes, there is a strong conflict between supporting high-throughput connections and the resulting over-sizing for slower connections. Reserving buffers for these oversized windows can be prohibitively expensive. Over-subscription avoids this problem but can lead to unnecessary packet loss and reduced performance.

4 Application Tuning of Buffers

Many operating systems allow system-wide default buffer sizes to be set while letting individual applica-

Table 1: Window-size = Delay × Bandwidth

Link	Speed	Window Size	
		80ms	500ms
	(Mb/s)	(KB)	(MB)
56k	0.05	0.56	0.003
T-1	1.55	15.9	0.1
T-3	45	460.8	2.8
OC-3	155	1587.2	9.7
OC-12	622	6369.28	38.9
GigE	1000	10240	62.5

tions select buffers that are smaller or larger than the default.

The system defaults can be raised, but will be used for all connections, even those over low delay-bandwidth networks. The calculations shown above make it clear why this is wasteful.

A few applications allow the user to request a certain buffer size, but the user must know what size to ask for. This information is usually obtained by sophisticated users who can measure the round-trip time with ping and determine the bandwidth through consultation with network providers or with the use of tools such as Pathchar [3] or Pchar [7]. Once determined, the values are specific to certain hosts and are subject to change at any time.

4.1 Measuring path characteristics

Like Traceroute, Pathchar and Pchar use the IP time-to-live field to determine the intermediate nodes in a path. Starting with the nearest hop and working out, the bandwidth impact of each additional link is calculated by measuring the amount of time it takes packets of various sizes to traverse that link.

For each packet size, multiple measurements are taken. It is assumed that if enough packets are sent, it is probable that at least one of them will go through that link without being delayed in queues. The minimum time measured for each packet size is therefore used to approximate the bandwidth.

This lengthy process is repeated for each hop to determine the bandwidth of each link. To measure end-to-end bandwidth, the hop-by-hop measurements could be avoided, but the number of samples necessary is probabilistically identical. The overall process can take over an hour to complete.

4.2 Automated discovery and use

If applications were to automatically use the algorithms used by these tools, prohibitively large amounts of bandwidth would be used performing these measurements. Further, these tools measure link bandwidth rather than the moving target of available bandwidth.

The Web100 Project [8] has architect-ed a new discovery protocol [9] for intermediate hops to communicate their link characteristics. The implementation of this protocol in the Internet core would require its adoption by the major router vendors.

5 Over-subscription

Another option is to change the specified behavior of TCP receivers to allow the over-subscription of buffer space. As proposed in [4], the receiver's buffers are tuned to some very large maximum value. The sender's buffers, no longer constrained by small window advertisements from receivers, are then allocated to sending processes using fair-share algorithms.

There is no discussion in this proposal, however, regarding the potential for starvation caused by this over-subscription.

A receiver uses its buffers for the following purposes:

Filling Gaps: If a gap is detected in a sequence of received packets, those packets are buffered until the gap can be filled. If the gap was caused by packet loss, the buffers will be used until the sender successfully retransmits the packet. If the culprit was the reordering of packets in the network, the missing packet may arrive very soon.

Application Buffering: Once a complete sequence of packets has been received and acknowledged, they can be delivered, in-order, to the receiving application. Using the Berkeley Sockets API, the data must reside in kernel buffers until the receiving application reads it in.

If one or more connections use significant portions of

its available buffers, the receiver could be forced to drop some incoming data for lack of buffer space.

5.1 Receiver consequences

The TCP standard specifies that a receiver can never shrink the window size for an existing connection because by the time the sender receives notification of the smaller window size, it may have already sent more data than will fit in the new window. If, however, the receiver is not concerned with strict adherence to window sizes, it can accept this possibility.

If a receiver were to lower the advertised window size when memory starts to become scarce, it would generally be successful at reducing the upper bound of the speed at which the sender could transmit data. However, to avoid dropping packets and triggering congestion control mechanisms, it must be prepared to receive more data than will fit in the new window.

If the receiver drops packets because it does not have enough free buffer space to hold that data, the sender will assume that the data was lost due to a congested link in the network. Congestion control mechanisms will be used and the sender will slow-down its transmissions. This merging of flow control and congestion control could provide relief to the receiver, but effectively takes flow control out of the hands of the receiver.

5.2 Applicability of congestion control mechanisms

The amount of buffers used on the receiver is not caused by the sender's speed, but instead by slow receiving applications, packet loss, or reordering in the network. If one connection with a large window comes through a path with packet loss or significant reordering, it will end up using a significant amount of receive buffer space while the holes are filled in. If another connection comes through without loss or reordering, it may be starved from the few buffers that it needs by the connection that suffered loss.

The effects of window congestion may be much more long-lived than traditional, instantaneous congestion of bandwidth or router queues. Out of order data can be thrown away and will be retransmitted at a slower pace, but data that has already been acknowledged cannot be removed from the buffers until the receiving application is waiting for it. In the meantime, well-behaved connections will be penalized by this blocking application. In essence, acknowledging data is committing not only the instantaneous use of buffers, but also use of those buffers for an unforeseeable amount of time. Thus, the fundamental congestion control technique of instantaneous packet loss and subsequent multiplicative back-off cannot always be exercised for window congestion. An application that has acknowledged data in the window may starve other connections without incurring a penalty of its own.

5.3 Conclusions

Over-subscription has been proposed as a technique for increasing the size of windows advertised by receivers. It is not clear, however, that the consequences of this over-subscription are tolerable. Under stress, the fairness to receiving processes is particularly troublesome. More study of this area is clearly warranted. There is a potential for a away from strict flow-control, but the full implications of such a move are not well understood. In addition, the probability that these problem cases will occur goes up as high delay-bandwidth network connections become more common.

6 Dynamic Right-sizing

A more appealing solution would be the development of a low-overhead mechanism for automatically and dynamically determining the correct window size for a given connection.

6.1 Attributes of the right size

The basic requirement is that, provided there is sufficient available memory, the window advertised by the receiver should be as large as the minimum of the bandwidth-delay product of the network path.

In practice, the sender's congestion window [6] will constrain the transmission rate for at least the first portion of the connection. Thus there is no value in advertising a window larger than the number of bytes that the sender can send with its current congestion window.[2]

[2]This calculation is made by taking the congestion window's

If, the congestion window is larger than the bandwidth-delay product, and the sender makes use of that window, it may experience packet loss and/or additional queuing delay. In the case of packet loss, the congestion window will be dropped. Additional queueing delay may allow for more full link utilization, but is also likely to cause packet loss. For this reason it is both undesirable and controlled by the congestion window.

Thus the congestion window is both an upper-bound on the bandwidth that the sender will use and an approximation of the connections fair share of the bandwidth-delay capacity of the network. Accordingly, our initial requirement can be replaced by saying the receiver should, provided there is sufficient available memory, advertise a window that is large enough to not constrain the congestion window.

6.2 Estimating the sender's congestion window

A TCP header option could be used for the sender to explicitly convey the window size to the receiver, but the deployment issues associated with any such addition to the protocol are prohibitively complex for the sake of this paper. A less intrusive technique is preferable.

It is already standard practice for the sender to measure the round-trip time of the connection [10]. There are no provisions, however, for measuring the available end-to-end bandwidth of the network.

It is clearly trivial to measure the amount of data received in a fixed period of time and subsequently compute the *average* throughput over that period. However, the instantaneous throughput of a connection seen by a receiver may be larger than the maximum available end-to-end bandwidth. For instance, data may travel across a slow link only to be queued up on a downstream switch or router and then sent to the receiver in one or more fast bursts.

The maximum size of this burst is bounded by the size of the sender's congestion window and the window advertised by the receiver. The 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.

packet count and multiplying it by the TCP maximum segment size.

Thus, for any period of time that is shorter than a round-trip time, the amount of data seen 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. Thus the window may be larger than the lower-bound.

As mentioned earlier, the window used by the sender is the minimum of the receiver's advertised window and the sender's congestion window. As a result, measuring this minimum and making sure that the receiver's advertised window is always larger will let the receiver track the congestion window size.

6.3 Averaging over longer periods

Over a period of time (t) that is larger than the round trip time, the amount of data received (d) is still a function of the round-trip time (rtt). Assuming that the round-trip time is not fluctuating, the time (t) is some whole number (n) of round-trip periods plus a partial period:

$$t = n \cdot rtt + rtt_{partial}$$

If the sender is sending as fast as possible, the amount of data sent during a round-trip time will be equal to the window size. Under the same assumption, the data received is equivalent to n times the window size plus some amount of data received during the partial period:

$$d = n \cdot window + window_{partial}$$

By substituting the two equations and solving for the window size, it is shown that the lower bound of the window size is bounded as follows:

$$\frac{d \cdot rtt}{t} \leq window \leq \frac{d \cdot rtt}{t} + \frac{d}{t}$$

Since the application may not have been ready to send for the entire time period, the estimated window size may be smaller than the actual window. If each time interval was measured separately, the largest measured value would be at least as large as the average value over a longer time period. Rather than using an average, we therefore use the largest measured value for each individual round-trip interval. This decreasing lower-bound is used to estimate the largest window that the sender may use.

6.4 Sender's window

The previous discussion presents a method for a receiver to enlarge its window to match the size of the sender's window. In current practice, the sender's window is often the same static size as the receiver's. A functional solution is for the sender and receiver to work their window sizes up until they are sufficient to fill the bandwidth-delay product.

To maintain strict flow control, the sender should not send more data than the receiver claims it is ready to receive. These increases must consequently be initiated by the receiver. It has already been stated that the receiver will advertise a window larger than the measured window. Thus the receiver may increase its buffer allocations by tracking the receiver's advertisements. This technique will not cause the sender's allocations to be out of line with the useful window size for the particular connection.

6.5 Slow-start

In order to keep pace with the growth of the sender's congestion window during slow-start, the receiver should use the same doubling factor. Thus the receiver should advertise a window that is twice the size of the last measured window size.

6.6 Window scaling

TCP only has a 16-bit field for window size. Newer implementations support Window Scaling [1] in which a binary factor (between 0 and 14 bits) is negotiated at connection setup for each end-point. For the remainder of the connection, this is implicitly applied to all windows advertised by that end-point.

However, there will be a resulting loss of granularity in expressing window sizes. For example, if the maximum window scaling of 14 bits is used, window sizes of 1 gigabyte can be represented, but the granularity will be 16 kilobytes. This makes it impossible to safely advertise a window size of less than 16 kilobytes as anything other than 0. Consequently, the receiver should use a window that is substantially larger than the quantum specified by the scaling factor.

In order to support dynamically sized buffers, the receiver must request, at connection setup, a window scaling sufficient to represent the largest buffer size that it wishes to be able to use. It must balance this scaling with the ability to represent smaller window sizes.

7 Implementation

The algorithms described above were implemented in a Linux 2.2.12 kernel. To make the receive throughput measurements, two variables are added to the TCP for each connection. These variables are used to keep track of the time of the beginning of the measurement and the next sequence number that was expected at that time.

Upon the arrival of each TCP packet, the current time is compared to the last measurement time for that connection. If more than the current, smoothed, round-trip time has passed, the last sequence number seen (not including the current packet) is compared to the next sequence number expected at the beginning of the measurement. Assuming that all packets are received in order, the result is the number of bytes that were received during the period. If packets were received out of order, the number of bytes received may actually be more or less.

The receive buffer space is then increased, if necessary, to make sure that the next window advertised will be at least twice as large as the the amount of data received during the last measurement period.

7.1 Receive buffer

The Linux 2.2 kernel advertises a receive window that is half as large as the receive buffer. Since data may come from the sender in packets with payloads anywhere from 1 byte to the maximum segment size, the receiver can never quite be certain how much buffer space will be used for a given amount of receive window. Thus, the buffers are advertised in a binary-search style. In practice, large amounts of data should come in large packets with relatively low storage overhead for headers.

Since the right-sizing algorithm described above already keeps the buffer size twice as large as the maximum amount of data received during a round-trip time, the amount of buffer space allocated is actually four times the last measurement. This is clearly wasteful,

but a better algorithm for safely filling the buffers is a different topic of study.

In addition, the Linux kernel uses the `netdev_max_backlog` parameter to limit the number of received packets that haven't been processed yet. This value defaults to 300 and was smaller than the sender's congestion window during the experiments shown below. Due to the bursty nature of TCP and the ability of gigabit ethernet to deliver packets very fast, we believe that we were passing this limit and changed the value to 3000.

7.2 Timing granularity

The granularity of kernel timers in this Linux kernel was 10ms. Even if the round-trip time is 1ms, the measurement period could be around 10ms (or vice-versa). For local-area connections with low round-trip times around 1ms, the calculated amount of progress may be inflated by as much as factor of 10. To compensate, the amount of data received during a measurement is averaged as if the measured period was actually 10 round-trips. As discussed earlier, average measurements made over multiple round-trip periods may be smaller than those made over single round-trip periods.

7.3 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. 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.

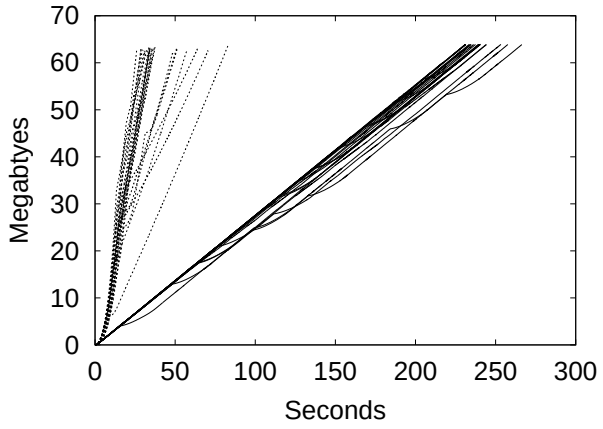

Figure 4: Progress of data transfers

To develop a useful upper-bound on the round-trip time, we keep track of the minimum value observed. If there is no smoothed, round-trip time (srtt) available from transmissions, the minimum received round-trip time is used.

8 Results

As expected, the use of larger windows increases performance compared to using smaller, default window sizes. In Figure 4, 50 transfers of 64 megabytes each were made with the `ttcp` program. 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. Both end points have gigabit ethernet interfaces separated by a WAN emulator that introduces a 100ms delay in the round-trip time. The congestion control mechanisms triggered by packet losses cause the abrupt decreases in slope.

8.1 Performance

The transfers made with default window sizes took a median time of 240 seconds to complete. The transfers with dynamic windows sizes were roughly 7 times faster and took a median time of 34 seconds.

In Figures 5 and 6, 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

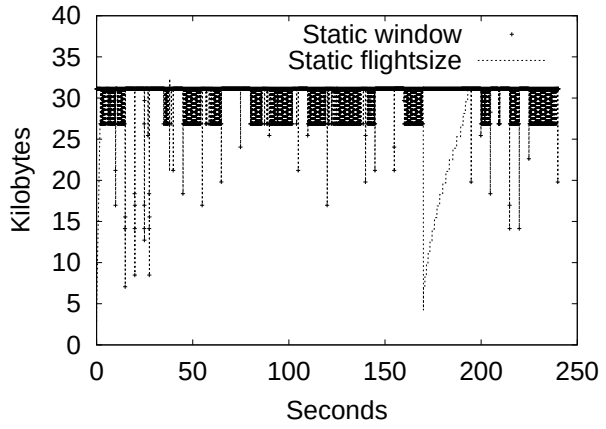

Figure 5: Static case: Flight & window sizes

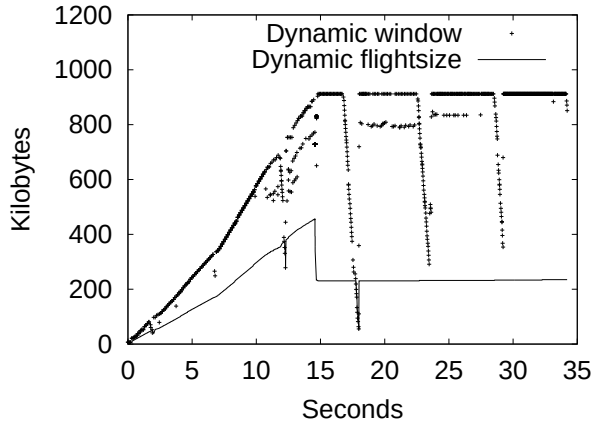

Figure 6: Dynamic case: Flight & window sizes

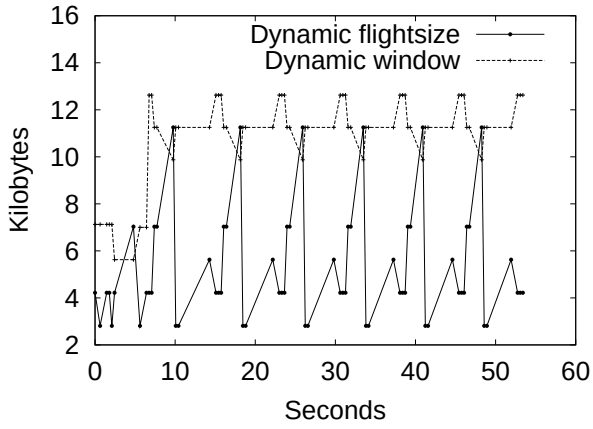

Figure 7: Low-bandwidth links: dynamic case

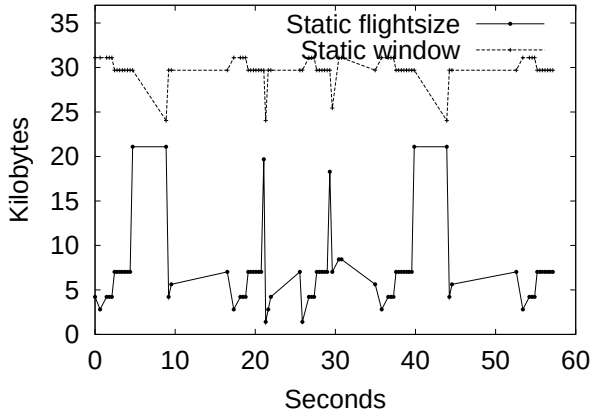

Figure 8: Low-bandwidth links: static case

the window advertised by the receiver.

Figure 6 shows that, in the default case, the flightsize is generally equal to the constraining window size. In contrast, during the dynamic case shown in Figure 5, 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. This 7-fold increase in the average flightsize is the source of the throughput increase demonstrated in Figure 4.

Other tests did occasionally see the increased queuing delay caused when the congestion window grows larger than the available bandwidth. At this point the retransmit timer expired and reset the congestion window even though the original transmission of the packet was acknowledged shortly thereafter. In this case the congestion window is reset to 1 rather than just performing a normal multiplicative decrease.

8.2 Low-bandwidth links

Figure 7 shows the first part of the same transfer over a link that is simulated to be only 56 kilobits. Here we see that the largest advertised window is under 13 kilobytes.

Figure 8 shows what happens over the same link when default window sizes are used. Other measurements show that the two cases get virtually identical throughput. Yet, the static case appears to usually have more data in flight.

As evidenced by the increased frequency of retransmissions show in Figure 9, this additional data in flight is dropped by the link which cannot support that

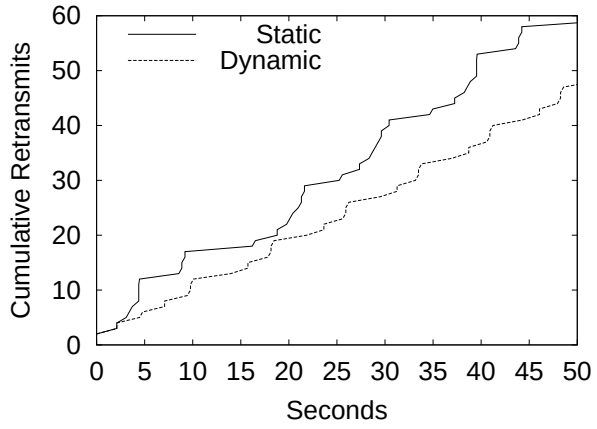

Figure 9: Low-bandwidth links: Retransmits

throughput. If the gateway to that link had sufficiently large buffers, the packet would be lost. If all network connections are behaving similarly, however, it is unlikely that such a gateway would possess enough buffer space for all connections.

9 Conclusions

The capability for scaling TCP's flow control windows has existed for several years, but the use of this scaling has remained dependent on manual tuning. To address this problem, we have developed a general method for automatically scaling the flow control windows in TCP. This scalability allows end systems to automatically support increased performance over high delay-bandwidth networks.

It has been demonstrated that this method can successfully grow the receiver's advertised window at a pace sufficient to avoid constraining the sender's throughput. For the particular circumstances tested a 7-fold speedup was achieved. Connections capable of supporting higher throughput would be able to realize their full throughput as well.

Meanwhile, the receiver remained in full control of the flow-control mechanisms. As a result, implementations that wish to guarantee window availability have the necessary information to strictly allocate buffers or control the degree to which they are over-committed. Additionally, network connections with small bandwidth-delay products are identified and the receiver can avoid reserving unnecessarily large amounts of buffer space for these connections.

At the same time, the sender's window also grows to match the congestion window. Fairness algorithms such as those described in [4] can be used to manage competition for sender buffer space. Further, these algorithms may also be useful for managing the receiver's buffer space.

Because the sender's window is throttled by the receiver's measurements of achieved throughput, the growth of the congestion window can be limited by actual performance. Over low-bandwidth bottlenecks, this feedback was able to reduce overall packet loss by keeping the congestion window closer to the available bandwidth.

In summary, we have shown that TCP flow control can be automatically scaled to support high delay-bandwidth links while providing more fairness than alternative solutions in which the receiver ignores actual flow control needs. In addition, more information is provided to the end-systems about actual flow control needs.

References

[1] V. Jacobson, R. Braden, and D. Borman, "RFC 1323: TCP extensions for high performance," May 1992.

[2] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "RFC 2018: TCP selective acknowledgment options," Oct. 1996, Status: PROPOSED STANDARD.

[3] Van Jacobson, "Characteristics of internet paths," Presentation published at ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf, Apr. 1997.

[4] J. Semke, J. Mahdavi, and M. Mathis., "Automatic TCP buffer tuning," in *Proceedings of ACM SIGCOMM '98*. 1998, pp. 315–323, ACM Press.

[5] Dan Kegel, "The C10K problem," Whitepaper published at http://www.kegel.com/c10k.html, 1999.

[6] J. Postel, "RFC 793: Transmission Control Protocol," Sept. 1981.

[7] Bruce Mah, "pchar: A tool for measuring internet path characteristics," http://www.employees.org/ bmah/Software/pchar/.

[8] "Web100 project," http://www.web100.org/.

[9] "Automatic bandwidth delay product discovery," http://www.web100.org/papers/bdp.discovery.html.

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