

Dynamic Right-Sizing: A Simulation Study

Eric Weigle and Wu-chun Feng
 {ehw, feng}@lanl.gov

Research & Development in Advanced Network Technology (RADIANT)
 Los Alamos National Laboratory
 Los Alamos, NM 87545

Abstract—Virtually all network applications requiring reliable end-to-end communication depend on TCP. Unfortunately, the performance of any stock TCP is abysmal over wide-area networks (WANs) and even over local-area networks (LANs) with very high-bandwidth links. Currently, network researchers manually optimize TCP buffer sizes to achieve acceptable performance over a given connection. Unfortunately, this manual optimization requires changes to the kernel on both end hosts involved in the network connection (changes that are only effective for connections between these two hosts). Furthermore, because two administrative domains must be coordinated to perform this optimization, this process can be tedious and time consuming.

To address these problems, this paper illustrates the benefits of a new technique called dynamic right-sizing. This technique dynamically and automatically determines the best buffer size, and hence flow-control window size in TCP. Our simulation study shows that dynamic right-sizing can improve the performance of flows by two orders of magnitude over stock TCP implementations that have static flow-control windows.

Keywords: dynamic right-sizing, high-performance networking, TCP, flow control, wide-area network.

I. INTRODUCTION

Over the past decade, TCP has become the ubiquitous transport protocol for the Internet.¹ However, stock TCP performs abysmally over high-bandwidth or high-delay links. As a result, the performance of application infrastructures such as computational grids [1] and high-volume web servers, which are built on TCP, is crippled.

To address this problem, grid and network researchers continue to manually optimize buffer sizes to keep the network pipe full, and thus achieve acceptable performance [2], [3]. Although such tuning can increase delivered throughput by an order of magnitude, it requires kernel-configuration changes that cannot be made by the end user. Instead, system administrators at the source and destination hosts must separately configure their systems to use large buffers, a tedious and time consuming process. Furthermore, manual tuning only works for the pair of hosts that are tuned. We propose a straightforward modification to TCP that automatically and transparently addresses the above problems while maintaining connection semantics and the ubiquitously deployed features of TCP.

To this end, we first briefly discuss TCP, focusing on its flow- and congestion-control mechanisms and problems with current implementations. We then discuss our approach and its implications, followed by our experiments and results. We finish with related work and conclusions.

¹While there are many versions of TCP, we focus on TCP Reno as it is the most commonly used and widely deployed variant. Hereafter, we mean *TCP Reno* when we refer to *TCP*.

A. TCP Flow and Congestion Control

TCP relies on two mechanisms, flow and congestion control, to set its transmission rate. Flow control ensures that the sender does not overrun the receiver's available buffer space while congestion control ensures that the sender does not unfairly overrun the network's available bandwidth. TCP implements these mechanisms via a flow-control window (*fwnd*) and congestion-control window (*cwnd*).

Specifically, TCP calculates an effective window (*ewnd*) as $ewnd \equiv \min(fwnd, cwnd)$ and then sends data at a rate of $ewnd/RTT$, where *RTT* is the round-trip time of the connection. While the *cwnd* varies dynamically as the network state changes, *fwnd* has always been static. Ideally, *fwnd* should vary with the bandwidth-delay product of the network.

B. The Failure of TCP

Historically, a static *fwnd* sufficed as all communication occurred over networks with low bandwidth-delay products. Setting *fwnd* to small values allowed acceptable performance while wasting little memory. Today, most operating systems set $fwnd \approx 64KB$, the largest window available without scaling [4]. Yet bandwidth-delay products range between a few bytes ($56Kbps \times 5ms \rightarrow 36B$) and a few megabytes ($622.08Mbps \times 100ms \rightarrow 7.8MB$). In the first case, we can waste over 99% of memory allocated ($36B/64KB = 0.05\%$). In the latter case, we waste 99% of the network bandwidth ($64KB/7.8MB = 0.80\%$).

During the life of a connection, network delay changes (due to transitory queueing and congestion) implying that the bandwidth-delay product also changes. Thus a fixed value for *fwnd* is never ideal; selecting a fixed value forces an implicit decision to (1) underallocate memory and underutilize the network or (2) overallocate memory and waste system resources. Clearly, the solution is to dynamically and transparently adapt *fwnd* to achieve good performance without wasting network or memory resources.

TCP suffers from problems other than the static *fwnd*, including self-similar (chaotic) behavior [5], [6], [7] and a slowly-converging, additive-increase period.² These congestion-control problems are serious and will appear in our results, but they are orthogonal to the problem that we addressing here, namely static flow-control windows. However, we are working on solutions to congestion control, as are others [8], [9], [10].

²For a 8MB bandwidth-delay product, convergence to the optimal bandwidth can take as long as $(8MB \times \frac{1}{2}) \times 1024^2 \frac{B}{MB} / 1500 \frac{B}{seg} \times 1 \frac{seg}{RTT} \times 0.100 \frac{s}{RTT} = 266s$, or nearly four and a half minutes!

II. DYNAMIC RIGHT-SIZING

Ideally, the transmission window ($ewnd$) should be limited only by the congestion window ($cwnd$). Recall that $ewnd \equiv \min(fwnd, cwnd)$, so network utilization is maximized when $fwnd \geq cwnd$, and memory use is most efficient when $fwnd = cwnd$.

For simulation purposes, we could simply set $fwnd = cwnd$. In our implementation, however, we provide some leeway ($|cwnd - fwnd| < \delta$) due to the nonzero overhead of memory allocation and deallocation. For δ sufficiently small, we can reproduce the performance of dynamically sizing flow-control windows in these simulations.

We refer to the process of dynamically sizing flow-control windows as *dynamic right-sizing* [11] and use the algorithm below to implement it.

Let $awnd \equiv$ receiver's advertised window (which is used to set $fwnd$ at the sender)

$cwnd \equiv$ sender's congestion window

$maxwnd \equiv$ maximum amount of memory that any one connection can utilize.

At the source,

$$fwnd = \min(cwnd, awnd, maxwnd).$$

At the destination,

$$awnd \geq 2 \times cwnd \text{ if sender in slow start}$$

$$awnd \geq cwnd + 1 \text{ if sender in additive increase.}$$

We modify the sender only by adding the parameter $maxwnd$, limiting the sending rate when administrators are unwilling to dedicate the resources required. With this formula, $maxwnd$ can be viewed as the maximum flow window or the maximum congestion window. We use $maxwnd$ to simulate the problem of sharing limited memory among multiple connections. In practice, it is unlikely to be used; instead a fairness algorithm would control memory allocation among competing connections.

The receiver must infer the TCP state and congestion window of the sender. This can be done by observing time stamps on received packets and calculating mean transfer rates over the last several RTTs. To conserve memory, $awnd$ can be reduced a few RTTs after a multiplicative decrease is detected (enough time for packets buffered in the network to reach the receiver).

A. Benefits of Dynamic Right-Sizing

The major benefits of dynamic right-sizing include improved memory and network performance. Our approach also gives full and valid interoperability with other TCP implementations and transparency to the user (i.e., no complex administration) and is also provably fair.

Other approaches either affect TCP semantics [12] or are specialized non-TCP protocols that are obviously not interoperable, particularly over the wide-area network (WAN), e.g., VIA [13]. For significant performance improvements, TCP implementations require the window-scaling extensions [4], but this technique works without them.

Often, system administration is just an irritating inconvenience, but sometimes it is more than that. When firewalls are

involved, connections between hosts on opposite sides are often broken into two connections — one from the source to the firewall and another from the firewall to the destination. So, regardless of the settings at the hosts, the firewall will be the bottleneck. Because firewalls are typically controlled by security-minded administrators, changing the firewall configuration can be difficult. If the firewall is under the control of a different institution, it may simply be impossible. Using dynamic right-sizing (as part of TCP) in the firewall makes this problem disappear.

Our approach is fair with respect to TCP Reno; we implement dynamic right-sizing *in* TCP Reno and do not alter its additive-increase, multiplicative-decrease (AIMD) mechanism. Specialized protocols like VIA cannot make this claim. Although flows using dynamic right-sizing acquire more bandwidth than those without it, that does not make it unfair. Flows that properly configure both endpoints *should* receive better performance than misconfigured flows. Dynamic right-sizing automatically makes the proper adjustments to flow-control windows, so we expect better performance with dynamically right-sized flows than for generally misconfigured static flows.

More rigorously, we use the following measure of fairness: *Flows should acquire bandwidth proportional to the resources (in our case, memory) they dedicate to the connection.* Thus, even if dynamic right-sizing reduces the share of bandwidth utilized by static flows, dynamic right-sizing would still be fair (since all the static flows would have to do to compete is to increase the memory allotted to their flow-control windows).

III. EXPERIMENTS

In this study, we simulate and analyze three general cases that show the effects of incremental adoption of dynamic right-sizing:

- All connections use static flow-control windows, i.e., today's Internet.
- Some connections use static flow-control windows while others use dynamic flow-control windows.
- All connections use dynamic flow-control windows.

Figure 1 shows the generic topology over which we run our simulation experiments. Links run at 100Mbps with 10ms delay in one set of experiments; in the other 1Gbps with 16ms delay. The RTTs for these experiments are 60ms and 96ms, respectively, representative of physically long connections present in the global Internet.

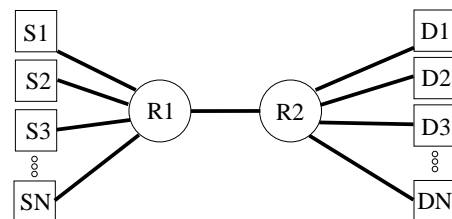


Fig. 1. Generic Topology

Buffer space available in network queues heavily influences TCP Reno's performance. This is due to the bursty nature of the traffic generated by TCP Reno [6], [7] causing dramatic variation in queue lengths. To study these effects, we simulate queue

sizes of 100, 500, and 16,384 packets in routers R1 and R2. The latter queue size represents an ideal router where packets are never dropped. Host buffers are set to not drop packets locally. All nodes used a drop-tail queuing algorithm; use of random early detection (RED) gateways did not noticeably change the overall results.

For static flows, we set f_{wnd} to the default of 64KB. All flows are one-way, constant bit rate (CBR) transmissions at the link speed (100 Mbps or 1000 Mbps) from sources S_i to destinations D_j where $i, j \in [1, n]$. Use of other traffic models such as Poisson or Pareto, or the inclusion of reverse path traffic [14] did not produce significantly different results. We believe that it is unlikely that traffic patterns alone would cause significant differences between TCP with and without dynamic right-sizing.

IV. RESULTS AND ANALYSIS

The experimental results in this section will show that TCP flows with dynamic right-sizing outperform flows without it. As a uniform basis for comparison, we track the delivered bandwidth at the receiver (as measured by the number of acknowledgement packets returned to the sender) as a function of max_{wnd} . We also consider performance with respect to link utilization, packet loss, and fairness among flows.

We structure our discussion as follows. We first discuss the current situation on the Internet where all flow-control windows are static in size, i.e., static flows. We then consider a head-to-head comparison of static flows vs. dynamic flows, i.e., dynamically right-sized flows. Lastly, we evaluate situations that would exist with incremental adoption of dynamic right-sizing.

A. Static Only

Over the 100Mbps topology, at most 1,666,666 packets can be sent during our 200-second simulation. This is shown as the maximum value on the y-axis in Figure 2. Each point in these figures represents an independent TCP simulation for the given maximum congestion window. The use of the line segments to connect the points represents the expected continuity when simulations are run at intermediate points (which we validated by several tests). For the 1000Mbps simulation, the value is ten times larger.

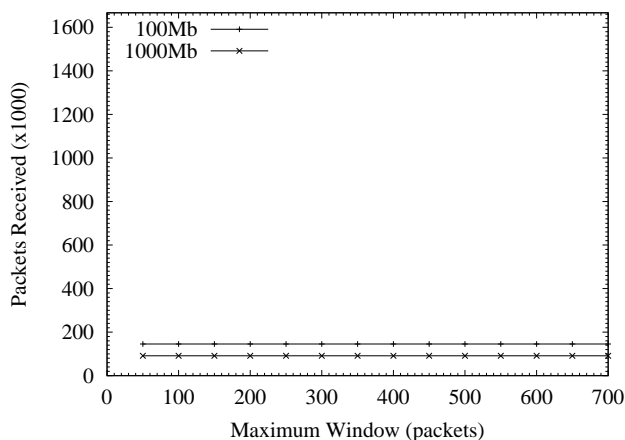


Fig. 2. Static Flow Only

The results in Figure 2 clearly illustrate the inadequacy of static flow-control windows in a WAN. In this case, the static flow throttles the bandwidth to $\frac{145,527}{1,666,666} \approx 8.7\%$ and $\frac{91,407}{16,666,666} \approx 0.55\%$ of what is available over the 100Mbps and 1000Mbps topologies, respectively! This is obviously atrocious and motivates the current laborious practice of hand-tuning connections. Using dynamic right-sizing would automatically increase f_{wnd} , and hence, $ewnd$ appropriately, without the waste of memory that manual tuning would incur.

The above behavior is clarified by the following analysis. With a fixed 64KB f_{wnd} (i.e., 44 standard Ethernet packets), $ewnd$ cannot grow beyond 64KB even though $cwnd$ can. Thus, the performance of the static flow is uniform no matter how large $cwnd$ grows beyond 64KB. And with the bandwidth-delay products of the 100Mbps and 1000Mbps networks being at least 500 and 5000 packets, respectively (more with realistic queueing delays), the static f_{wnd} throttles bandwidth to at most $\frac{44}{500} \approx 8.8\%$ and $\frac{44}{5000} \approx 0.88\%$ of what is available over the 100Mbps and 1000Mbps topologies, respectively. These percentages decrease further with increased bandwidth-delay products caused by realistic queueing delays.

B. One Static vs. One Dynamic

Figure 3 presents results for one flow of each type and queue sizes of 100 and 500 packets over the 100Mbps network. We vary f_{wnd} from 50 packets to 700 packets, sizes corresponding to window sizes of 73KB (i.e., just larger than the static flow window size of 64 KB) up to 1 MB. This case most clearly shows the interaction between flows using static and dynamic flow-control windows.

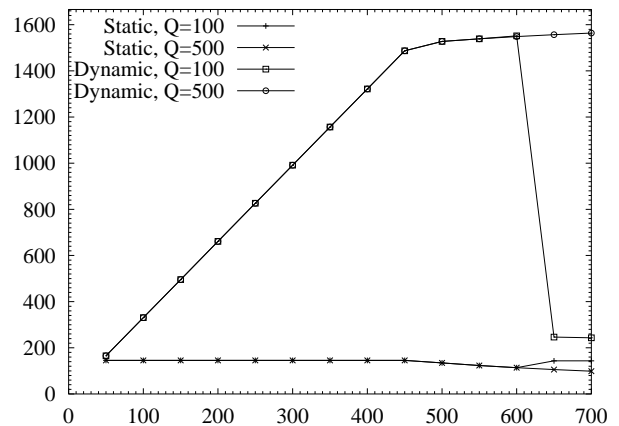


Fig. 3. Static vs. Dynamic Flow

The upper two lines are dynamic flows while the lower two are static flows. When the rate limiting max_{wnd} is relaxed, the dynamic flows deliver over 90Mbps throughput, while the static flows remain unaffected and deliver a paltry 8.7 Mbps of throughput!

No differences based on queue size are evident until max_{wnd} reaches 650 packets. There, for a queue size of 100 packets, the dynamic flow overflows the router's buffer and induces massive packet loss. The dynamic flow cuts its sending rate due to these

losses, and the static flow regains some bandwidth.

This value ($maxwnd \approx 650$) is significant; it is precisely the actual bandwidth-delay product for this network with queueing delays. A study of $cwnd$ over time for queue size 100 and a $maxwnd = 650$ explains the performance drop.

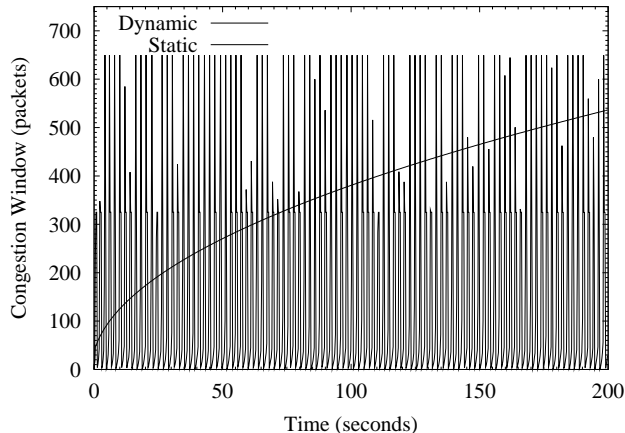


Fig. 4. Congestion Window Over Time

While the graph of $cwnd$ for the static flow is smooth curve, it is highly variable for the dynamic flow. This occurs due to the use of slow-start restart. The dynamic flow enters slow start at the beginning of the transmission, allows its congestion window to grow up to the limit of 650 packets, which then allows a large burst onto the network, and produces loss rates of up to 27%. The dynamic flow interprets this as a sign of heavy congestion and temporarily backs off completely, but when it attempts to continue transmission, it again uses the slow-start mechanism, and the cycle continues.

This is a problem with TCP and the AIMD mechanism. If we turned off slow-start restart, performance would improve, but it would be a point-specific solution. In many cases, slow-start restart actually helps performance. This problem is unfortunately brought to the forefront by the use of dynamic right-sizing. Note that hand-tuning these connections would do even worse as a window must be selected before the transient connection dynamics are known. Furthermore, even the "bad" part of this graph for flows with $maxwnd \geq 600$, using dynamic right-sizing still performs significantly better.

We propose three solutions to the above problem. The first solution simply increases the queue sizes in the routers so that losses are simply avoided, the effects of which are seen in Figure 3 for a queue size of 500 packets. In this case, the dynamic flow achieves nearly 100% network utilization and delivers 93.8Mbps throughput! However, this solution is the least realistic of the three solutions we discuss here because the Internet is an immense, decentralized, distributed system. The likelihood of all the routers in the Internet being upgraded with large buffers is effectively zero; and in fact, vendors generally under-engineer their routers with less buffer space and rely on TCP for retransmissions in order to save money.

Another solution resets the congestion window to various fractions of the prior window after a loss [15]. This approach is quite important when the network delay is high because ad-

ditive increase takes too long to converge (see section I-B) over high bandwidth*delay networks. Even better would be to avoid the losses altogether as other versions of TCP such as TCP Vegas [16] would do.

Our last solution is probably the most realistic and deployable of the three: adding a simple heuristic over the $maxwnd$ variable that collects performance information based on the value of $maxwnd$. If performance degrades when $maxwnd$ is increased and the congestion window is fluctuating rapidly, we know that we are overshooting available bandwidth and need to decrease $maxwnd$ slightly. In essence, this technique uses the flow-control mechanism as a secondary congestion-control mechanism, which is inelegant but effective [10].

For brevity, the results over the 1000Mbps network are not included here as they are similar to the 100Mbps results but with appropriate scaling by a factor of ten with respect to bandwidth, network utilization, and optimal "maximum window" size, i.e., 6500 packets for queue size 100.

C. Mostly Static

Figure 5 shows the results of our experiments with ten static TCP flows and one dynamic TCP flow. Two sets of lines are plotted: one representing the arithmetic mean of the static flows and the other for the single dynamic flow. The scale on the dependent axis has been reduced from 1,666 to 1,200 thousand packets as the increased number of flows individually acquire less bandwidth. Three router queue sizes are tested: 100, 500, and 16,384.

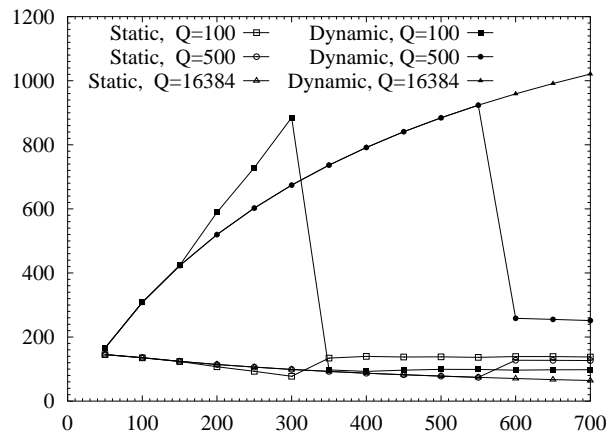


Fig. 5. Ten Static Flows, One Dynamic Flow

We observe similar behavior as in the prior section. For the smallest queue size, there is a sudden performance drop as we overflow the queueing available in the network, when $maxwnd > 300$. The value here is lower than the prior section due to the higher number of flows competing for queue space. Unfortunately, this is also a situation where static flows outperform dynamic flows. With 11 competing flows, the effective queue size per flow is only 9 packets; large windows permit TCP Reno to send large bursts, and thus incur many losses. Recall that this is a known problem of TCP Reno with known solutions.

For a queue size of 500, the network can handle larger bursts. Here, the dynamic flow performs well until $maxwnd = 600$. At this point, since the effective queue size per flow is now 45 packets, the dynamic flow still outperforms the static flows.

Lastly, with a queue size of 16,384 (large enough to avoid packet loss), the dynamic flow achieves nearly 16 times the bandwidth of any static flow. For $maxwnd = 700$, network utilization is over 99%.

As shown in Figure 5 for queue sizes of 500 and 16,384, the dynamic flow demonstrates a sublinear increase in bandwidth as $maxwnd$ increases; an increase in $maxwnd$ leads to larger bursts, that leads to longer queueing delays, that leads to an increased bandwidth-delay product, that in turn requires an even larger flow window for full network utilization. Increased delay also causes decreased performance for static flows as they hold constant the number of packets allowed outstanding per RTT while the RTT increases from 60ms to over 120ms in this case. Other TCP versions, such as TCP Vegas, avoid this feedback behavior by avoiding the use of large amounts of queue space in the network.

As far as the network is concerned, dynamic flows are no more than static flows with large buffers allocated. Thus, all prior research into the fairness and stability of TCP carries over when dynamic right-sizing is used. Although the dynamic flow achieves larger bandwidth than the static flows for sufficiently large $maxwnd$, this is not unfair. To show this, we use our measure that flows should achieve bandwidth comparable to the resources they dedicate to the connection. The dynamic flow allocates 16 times the memory (700 packets = 1 MB versus 44 packets = 64 KB) and receives 16 times the bandwidth. Under this measure, the results above are perfectly fair for a queue size of 16,384 and quite unfair against dynamic flows for a queue size of 100.

For the 1000Mbps network, we find results similar to Figure 5 with $maxwnd$ and queue sizes appropriately scaled up by a factor of ten.

D. Equal Number of Static & Dynamic

Figure 6 shows the results of our experiments with an equal number of dynamic and static flows, five each. As in the previous section, we plot two curves: one representing the mean throughput of the static flows and the other representing the mean throughput of the dynamic flows. The scale on the dependent axis is reduced further from 1200 to 400 packets as the increased number of flows each acquire less total bandwidth.

For the smallest queue size, we see some erratic behavior. As in the "mostly static" set of experiments with a queue size of 100, the performance of dynamic flows drops heavily for $maxwnd \approx 400$ when queues become full. Unexpectedly, there is another jump when $maxwnd = 650$. A closer look at the pre-averaged data reveals an interesting anomaly that we address below.

Table I shows that the jump in the averaged data is due to a single dynamic flow's acquisition of many times the bandwidth of other dynamic flows. Unfortunately, the explanation is due to our use of a deterministic simulator and constant packet size. When packets arrive at a router on multiple links simultaneously, they are queued based on the numbering of the links,

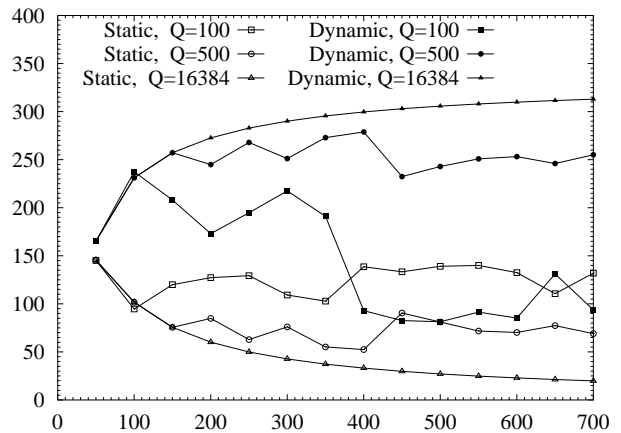


Fig. 6. Five Static Flows, Five Dynamic Flows

Flow #	1	2	3	4	5
Static	108	112	113	112	108
Dynamic	389	66	55	71	74

TABLE I
THOUSAND PACKETS ACKNOWLEDGED FOR 100MB
NETWORK, Q=100, MAXIMUM DYNAMIC WINDOW=650

i.e., lowest number first. Thus, when packets arrive at the same time, certain flows (those on lower numbered links) will have an advantage — especially when the first packet enqueued fills the queue and the second one is dropped.

In an attempt to minimize the effects of deterministic queueing, we interleaved the creation of static and dynamic flows. Unfortunately, that was insufficient here due to the particular packet timings. Similar artifacts can be seen in the real world, based on the particular queueing disciplines of routers. Random queueing order and random packet sizes reduce this effect but produce other unusual patterns.

When the queue size is 500, we again find the expected performance although it is slightly obfuscated due to the larger variation that five dynamic flows allow. The point at which network queues become saturated is less noticeable but still apparent at $maxwnd \approx 450$.

For the largest queue size, behavior is also similar to this case in the prior section. Comparing this graph to the graph with ten static and one dynamic flows for $maxwnd = 700$, we see that the drop in each static flow's performance is now closer to $\frac{5}{6}$ instead of $\frac{1}{2}$. That might seem unfair. However, we verify the experimental results in Figure 6 using our fairness definition.

- 10 flows share bandwidth, 5 of which allocate $700 \text{ segments} \times 1500 \frac{\text{bytes}}{\text{segment}} = 1 \text{ MB}$ of memory and 5 of which allocate 64 KB of memory, for a total of 5440 KB allocated.
- Assuming 100% link utilization, a flow should receive $\frac{100 \text{ Mbps}}{5440 \text{ KB}} = 18.38 \text{ Kbps}$ per kilobyte they allocate to the connection.
- Thus, dynamic flows should acquire $\frac{18.38 \text{ Kbps}}{1 \text{ KB}} \times 1024 \text{ KB} = 18,820 \text{ Kbps}$, while static flows should acquire $\frac{18.38 \text{ Kbps}}{1 \text{ KB}} \times 64 \text{ KB} = 1,176 \text{ Kbps}$.

- That resolves to 314,000 packets for dynamic connections and 19,600 packets for static connections, given a 200s simulation and 1500B segments.
- These values correspond almost exactly to the last graph in Figure 6.

Similar analysis can be performed for the remaining points to conclude that the performance differences between static and dynamic flows is fair.

Because the results over the 1000Mbps topology are somewhat different for this set of experiments, Figure 7 present the results here for five static and five dynamic flows at 1000Mbps speeds and queue sizes of 100 and 16,834. The scale has been increased by a factor of ten.

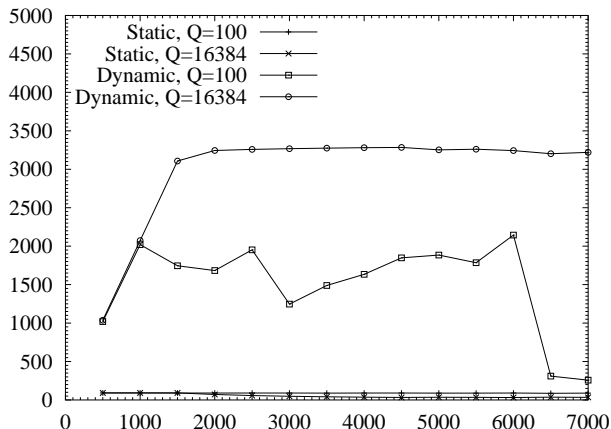


Fig. 7. Five Static, Five Dynamic at High Bandwidth

For the queue size of 100, we see the same expected drop when queues are saturated for a maximum window of 6500 packets. Increasing the queue space to 16,384 packets again increases the performance of the dynamic flows dramatically and decreases the bandwidth acquired by the static flows as dictated by the fairness measure. This graph shows how painfully inadequate static TCP flows are for high bandwidth connections with high delay. For a maximum dynamic window of 7000 packets (10MB), the dynamic flow utilizes 93 times the bandwidth of a static flow! Ten static flows would utilize less than 6% of the available bandwidth while the five static and five dynamic flows utilize about 99.5% of the link’s bandwidth for all maximum dynamic windows at or above 2000 packets.

E. Mostly or All Dynamic

When most flows are dynamic, the effects the few static flows have on the network are negligible. Consequently, this case is essentially that of all dynamic flows. We combine our discussion of the two here.

Thus far, we have presented results showing how well dynamic flows perform over static flows. One foreseeable problem is that when a network runs only dynamic flows, it may be less stable due to heightened competition between flows. This does occur but produces behavior no more adverse than when too many static flows try to utilize a link.

Figure 8 shows the results from our experiments with ten dynamic flows on the 100Mbps network with queue sizes of 100

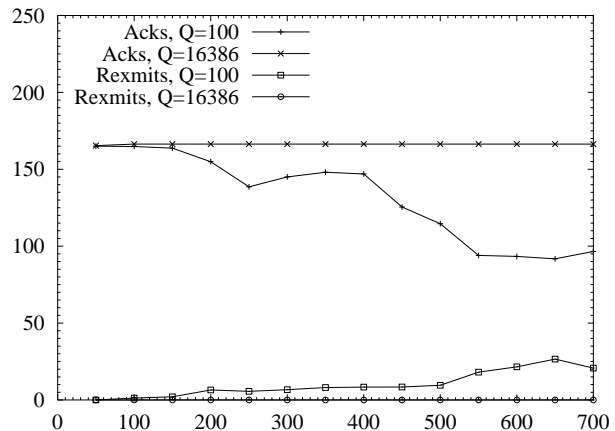


Fig. 8. Ten Dynamic Flows

and 16,384. When the queue size is 100, the large losses are caused by the slow start / slow-start restart problem that we discussed earlier. Increasing the queue size decreases these losses to zero, and we see a perfectly horizontal line where each flow shares the same bandwidth. This can be explained as follows: For this 100Mb network, the fair bandwidth per flow is 10Mb. This leads to a fair bandwidth-delay product of between 50 and 100 packets, depending on queueing delay. This value is below the maximum value on the x-axis of this graph. Thus performance does not depend on *maxwnd* — all values in this graph are above 50 and never limit our flows.

V. RELATED & FUTURE WORK

Researchers at the Pittsburgh Supercomputing Center have an implementation of a similar idea which they refer to as “Automatic TCP Buffer Tuning” [12]. Our approach differs from theirs in one significant aspect: their advertised window is always simply the maximum socket size. This assumes that memory on the receiver is never a concern. We believe that this is an invalid assumption and that it violates TCP semantics in the sense that there will be cases when the host is unable to accept as much data as its advertised window claims it is able to accept. One case where this is serious is the case where a server has many connections such that the sum of the advertised windows is greater than available memory. The receiver must either drop packets it advertised it could handle (violating TCP semantics) or try to allocate nonexistent memory (crashing the machine). The choice will obviously be to drop packets, which seriously affects performance due to TCP congestion-control mechanisms. Our approach avoids these problems by maintaining the TCP flow-control semantics and using the advertised window in its original sense.

There are many problems with TCP in its various incarnations, as we have repeatedly pointed out. Many of these [17], [6], [7] become more pronounced when dynamic right-sizing is used and make it more important to apply the known solutions [16], [18], [19], [20]. As system administrators and kernel developers slowly come to the realization that they need to upgrade their implementations of TCP, those problems will disappear.

In particular, we also performed all the above simulations

with TCP Vegas and found that with the proper choice of its α and β parameters, performance is generally better than TCP Reno, and there is less dependence on router buffer sizes. Using packet spacing with TCP Vegas eliminates the dependency altogether. We chose to present results for TCP Reno alone as few people use TCP Vegas.

We are currently working on an implementation of this work in the Linux 2.4.4 kernel and plan to investigate the effects of dynamic right-sizing when other TCP extensions such as Forward Acknowledgement [21] or TCP Pacing [17] are enabled. In addition, we will study how dynamic right-sizing interacts with routers that implement explicit congestion notification [22], [23] and new queueing disciplines such as BLUE [24].

VI. CONCLUSION

In this paper, we presented a comprehensive set of simulation results that demonstrate the benefits of using dynamic right-sizing in a wide range of networks. The approach delivers high performance and satisfies the goals of being dynamic and efficient with respect to utilizing resources as well as being fair and transparent to the end user, as described below.

First, by using an implementation of TCP that dynamically resizes its flow control window, the performance of TCP is no longer artificially throttled and can deliver orders of magnitude higher throughput between communicating nodes in a computational grid or to high-volume web servers. Second, this technique results in better overall network and kernel memory utilization by not statically allocating inappropriately sized buffers. Third, dynamic right-sizing eliminates the need for having to deal with the administrative headaches of optimizing buffers. Fourth, dynamic right-sizing is fair in the sense that flows acquire bandwidth proportional to the resources that it dedicates to the connection. And lastly, the above benefits are achieved transparently to the end user.

REFERENCES

- [1] Ian Foster and Carl Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1999.
- [2] Pittsburgh Supercomputing Center, "Enabling High-Performance Data Transfers on Hosts," http://www.psc.edu/networking/perf_tune.html.
- [3] Brian Tierney, "TCP Tuning Guide for Distributed Applications on Wide-Area Networks," in *USENIX & SAGE Login*, <http://www.didc.lbl.gov/tcpwan.html>, February 2001.
- [4] D. Borman, R. Braden, and V. Jacobson, "TCP extensions for high performance (RFC1323)," May 1992.
- [5] Wu-chun Feng and P. Tinnakornsriruphap, "The Failure of TCP in High-Performance Computational Grids," in *Proceedings of SC 2000*, November 2000.
- [6] András Veres and Miklós Boda, "The Chaotic Nature of TCP Congestion Control," in *Proceedings of IEEE Infocom 2000*, March 2000.
- [7] Y. Richard Yang, Min Sik Kim, and Simon S. Lam, "Transient Behaviors of TCP-friendly Congestion Control Protocols," ftp://ftp.cs.utexas.edu/pub/lam/transient_tech.ps.gz, July 2000.
- [8] Deepak Bansal and Hari Balakrishnan, "Binomial Congestion Control Algorithms," in *Proceedings of IEEE INFOCOM 2001*, April 2001.
- [9] Sergey Gorinsky and Harrick Vin, "Additive Increase Appears Inferior," Tech. Rep. TR2000-18, UT-Austin, May 2000.
- [10] Lampros Kalamopoulos, Anujan Varma, and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method To Enhance TCP Performance," Tech. Rep. UCSC-CRL-97-21, University of California at Santa Cruz.
- [11] Mike Fisk and Wu-chun Feng, "Dynamic Adjustment of TCP Window Sizes," <http://public.lanl.gov/mfisk/fisk/web/papers/tcpwindow.pdf>, July 2000.
- [12] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP Buffer Tuning," *ACM SIGCOMM 1998*, vol. 28, no. 4, October 1998.
- [13] Compaq Computer Corporation, Intel Corporation, and Microsoft Corporation, "Virtual Interface Architecture," <http://www.viarch.org/>.
- [14] Lixia Zhang, Scott Shenker, and David D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," in *Proceedings of ACM SigComm 1991*, September 1991.
- [15] Y. Yang and S. Lam, "General AIMD Congestion Control," in *Proceedings of ICNP 2000*, May 2000.
- [16] Lawrence Brakmo and Larry Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 8, pp. 1465–1480, October 1995.
- [17] Amit Aggarwal, Stefan Savage, and Thomas Anderson, "Understanding the Performance of TCP Pacing," in *Proceedings of IEEE INFOCOM 2000*, March 2000.
- [18] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas Revisited," in *Proceedings of IEEE Infocom 2000*, March 2000, pp. 1546–1555.
- [19] Richard J. La, Jean Walrand, and Venkat Anantharam, "Issues in TCP Vegas," http://www.path.berkeley.edu/hyongla/PAPERS/vegas_issue.ps.
- [20] Steven Low, Larry Peterson, and Limin Wang, "Understanding TCP Vegas: Theory and Practice," Submitted for publication, February 2000.
- [21] Matthew Mathis and Jamshid Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control," *ACM Computer Communication Review*, vol. 26, no. 4, October 1996.
- [22] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, October 1994.
- [23] K. Ramakrishnan and Sally Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP (RFC2481)," January 1999.
- [24] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue: A New Class of Active Queue Management Algorithms," University of Michigan CSE-TR-387-99., April 1999.
- [25] Van Jacobson, "Congestion Avoidance and Control," *ACM Computer Communications Review*, vol. 18, no. 4, pp. 314–329, August 1988.