Scalable TCP: Improving Performance in Highspeed Wide Area Networks

This work is submitted to the PFLDnet 2003 workshop for the purposes of discussion. The original paper was submitted in December 2002 to a journal for publication.

Tom Kelly
CERN* / University of Cambridge†
ctk21@cam.ac.uk

21 December 2002

Abstract

TCP congestion control can perform badly in highspeed wide area networks because of its slow response with large congestion windows. The challenge for any alternative protocol is to better utilize networks with high bandwidth-delay products in a simple and robust manner without interacting badly with existing traffic. Scalable TCP is a simple sender-side alteration to the TCP congestion window update algorithm. It offers a robust mechanism to improve performance in highspeed wide area networks using traditional TCP receivers. Scalable TCP is designed to be incrementally deployable and behaves identically to traditional TCP stacks when small windows are sufficient. The performance of the scheme is evaluated through experimental results gathered using a Scalable TCP implementation for the Linux operating system and a gigabit transatlantic network. The results gathered suggest that the deployment of Scalable TCP would have negligible impact on existing network traffic at the same time as improving bulk transfer performance in highspeed wide area networks.

1 Introduction

A communication network can experience periods where the traffic offered to it exceeds the available transmission capacity; during such periods the network is said to be *congested*. TCP congestion control [9] was introduced to relieve congestion collapse that had occurred in the Internet. A result of congestion control is that resources are shared between flows during periods of congestion. This sharing leads to similar throughput for flows with similar round trip times and avoids starving individual flows. TCP has proved to be remarkably successful at sharing bandwidth while agressively utilizing available capacity under a range of dynamic traffic loads.

The TCP flow control algorithm uses a window and end-to-end acknowledgment scheme to provide reliable data transfer across a network; a brief description is given here and a more complete reference is [15]. The sending host maintains a congestion window, *cwnd*, which places an upper bound on the number of segments that may be sent into the network awaiting acknowledgment by the receiver. Upon receiving a data packet the receiver schedules a cumulative acknowledgment, that covers all received packets, to be sent to the sender. The receiver also advertises to the sender a receive window, *rwnd*, which is the size of the available socket receive buffer for this connection. The sender is allowed to have at most the minimum of *cwnd* and *rwnd* packets in the network awaiting acknowledgment. The receive window provides flow control for the receiving application; if the receiving application cannot process data at the speed it is being sent the

*IT Division, CERN, 1211 Geneva 23, Switzerland.

†Laboratory for Communication Engineering, Cambridge University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, United Kingdom.

Throughput	Window	Packet loss recovery time	Supporting loss rate
1Mbps	17pkts	1.7s	5.2×10^{-3}
10Mbps	170pkts	17s	5.2×10^{-5}
100Mbps	1700pkts	2mins 50s	5.2×10^{-7}
1Gbps	17000pkts	28mins	5.4×10^{-9}
10Gbps	170000pkts	4hrs 43mins	5.4×10^{-11}

Table 1: Characteristics of a 200ms TCP connection using traditional congestion control.

window advertisements from the receiver, *rwnd*, will shrink as the socket receive buffer fills. The congestion window is intended to provide flow control during periods in which the network is congested. Packet loss is detected either through the timeout of an unacknowledged packet, the receipt of several duplicate acknowledgments, or through selective acknowledgment (SACK) reports [12] sent by the receiver. Packet loss is used as a signal of congestion; it is assumed to be caused by a buffer overflow due to offered traffic exceeding available capacity on the end-to-end path of a connection.[1] TCP senders update the congestion window in response to acknowledgments of received packets and the detection of congestion. For each acknowledgment received in a round trip time in which congestion has not been detected

$$cwnd \mapsto cwnd + \frac{1}{cwnd} \tag{1}$$

and on the first detection of congestion in a given round trip time

$$cwnd \mapsto \frac{cwnd}{2} \tag{2}$$

This process of increasing and decreasing *cwnd* allows TCP to aggressively utilize the available bandwidth on a given end-to-end path.

The agility of this congestion window adjustment algorithm can be studied by considering the time taken to reach the same sending rate following the detection of a transient congestion event. Suppose a connection has a round trip time of 200ms and a packet size of 1500 bytes. An available bandwidth of 1Gbps corresponds to a congestion window of about 16000. Immediately after the detection of a congestion event *cwnd* will be set to 8000, which is equivalent to sending at 500Mbps. To reach the sending rate of 1Gbps again will take 8000 round trip times or about 27 minutes! In many highspeed wide area networks this recovery time is much longer than the time between transient congestion periods. This can lead to low utilization even when the network is uncongested for extended periods.

However, by altering the congestion window adjustment algorithm, the agility with large windows can be dramatically improved. This paper will consider the use of the following congestion control algorithm. For each acknowledgment received in a round trip time in which congestion has not been detected

$$cwnd \mapsto cwnd + 0.01 \tag{3}$$

and on the first detection of congestion in a given round trip time

$$cwnd \mapsto cwnd - \lceil 0.125 * cwnd \rceil \tag{4}$$

The time taken for a source using this algorithm to double its sending rate is about 70 round trip times for *any* rate; the window update algorithm is *scalable* and a TCP implementing it is termed Scalable TCP. In the previous case of a 1Gbps connection with a round trip time of 200ms, the scalable algorithm will recover its original rate after a transient in under 3 seconds. This suggests

[1]The use of explicit congestion notification (ECN) [8] by routers allows congestion to be signaled to the sender (via acknowledgments from the receiver) without the loss of packets.

that this algorithm could better utilize the bandwidth of a highspeed wide area network that experiences transient congestion.

This paper studies the design, implementation, and presents early results on the performance of the Scalable TCP modification to TCP congestion control. Section 2 describes the problems associated with TCP congestion control in highspeed wide area networks and presents a context within which Scalable TCP would be beneficial. Section 3 considers the analytical properties of the generalized Scalable TCP algorithm and motivates the choice of the parameters 0.01 and 0.125. Section 4 presents results of experiments performed using a Scalable TCP implementation in the Linux operating system over the DataTAG highspeed transatlantic testbed. Section 5 considers how this scheme differs from the related work on improving the performance of congestion control in high speed networks. Section 6 summarizes what has been achieved and gives directions for future work.

2 Motivation and Context

This work is motivated by the poor performance of TCP when used for bulk transfers in highspeed wide area networks. These networks have speeds greater than 100Mbps and round trip times above 50ms. Several communities use such networks and need to distribute substantial amounts of data over them. For example, the large datasets collected by the High Energy Physics, Bioinformatics and Radioastronomy communities require global distribution for the data to be analyzed effectively.

Define the *supporting loss rate* for a connection to be the maximum packet loss rate that a congestion control algorithm will tolerate to sustain a given level of throughput. Let the *packet loss recovery time* for a given rate and connection be the length of time required by a congestion control algorithm to return to its initial sending rate following the detection of a packet loss. Traditional TCP connections are unable to achieve high throughput in highspeed wide area networks due to the long packet loss recovery times and the need for low supporting loss rates. Table 1 shows the properties of a traditional TCP connection with a round trip time of 200ms and a segment size of 1500 bytes. A packet loss rate of 10^{-7} is comparable with those that can occur on long haul fiber links, within network devices, and in end-systems; this places a limit on throughput before any transient congestion due to load fluctuations are considered. This constraint on the loss rate becomes problematic for a connection with a round trip time of 200ms at around 100Mbps. Furthermore the packet loss recovery time for a 10Mbps connection with round trip time of 200ms becomes comparable with inter-page think times for a user's Web requests. A recovery time of more than a few minutes could be detrimental to efficient utilization of a network with periods of transient congestion; at a round trip time of 200ms this effect would occur at rates of more than 100Mbps.

This paper considers whether a simple change to the congestion control algorithm is sufficient to improve highspeed wide area network operation. Scalable TCP is an evolution of the existing congestion control algorithm that improves performance when there is a high available bandiwdth on long haul routes. It is designed to be easily implemented in current TCP stacks and incrementally deployable without needing modifications to network devices. Scalable TCP builds on the HighSpeed TCP proposal [6] and previous work on engineering stable congestion controls [11].

3 Analysis and Design

The analysis will make use of standard fluid limit approximations and the following notation conventions. Let each source and destination pair in the network be identified with a route, r, and the end-to-end dropping probability on a route be denoted by $P_r(t)$. Let $cwnd_r$ and T_r denote the sender's congestion window and the round trip time of a connection on route r.

The generalized Scalable TCP window update algorithm responds to each acknowledgment received in a round trip time in which congestion has not been detected with the update

$$cwnd_r \mapsto cwnd_r + a \tag{5}$$

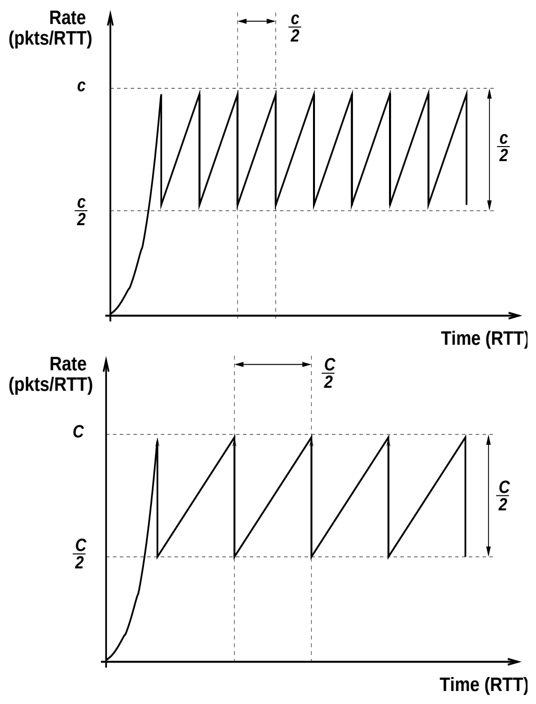

Figure 1: Traditional TCP scaling properties.

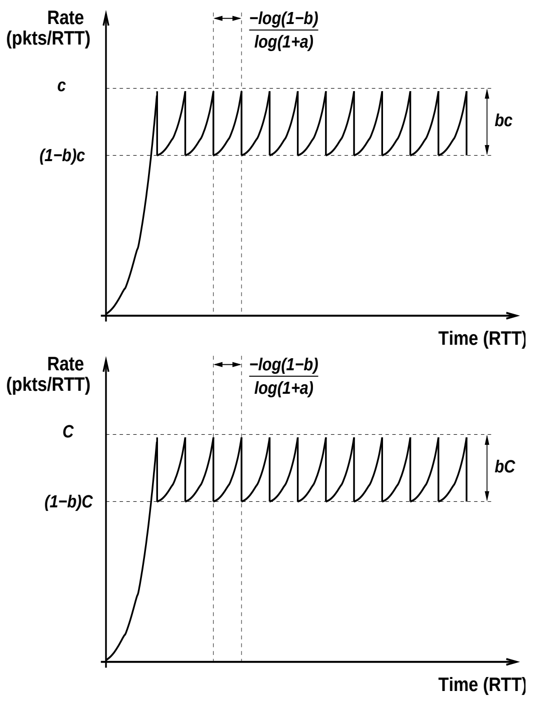

Figure 2: Scalable TCP scaling properties.

where a is a constant with $0 < a < 1$. Further, on the first detection of congestion in a given round trip time, the congestion window is altered by

$$cwnd_r \mapsto cwnd_r - \lceil b * cwnd_r \rceil \tag{6}$$

where b is a constant with $0 < b < 1$. Figures 1 and 2 illustrate the congestion window dynamics of a single connection using traditional TCP or Scalable TCP over a dedicated link of capacity c or C ($c < C$). Packet loss recovery times for a traditional TCP connection are proportional to the connection's window size and round trip time. A Scalable TCP connection has packet loss recovery times that are proportional to the connection's round trip time only; this invariance to link sizes allows Scalable TCP to outperform traditional TCP in highspeed wide area networks. The scaling property applies for any choice of the constants a and b; implementation and deployment constraints determine these constants. The use of $a = 0.01$ and $b = 0.125$ will be motivated by considering Scalable TCP's impact on legacy traffic, bandwidth allocation properties, flow rate variance, convergence properties, and control theoretic stability.

3.1 Response curve and bandwidth allocation

A congestion window update algorithm relates the congestion window size to the end-to-end signaling rate through a *response curve.* The generalized Scalable TCP algorithm has a response curve that can be approximated for small end-to-end drop rates by[2]

$$cwnd_r \approx \frac{a}{b}\frac{1}{P_r} \tag{7}$$

The traditional TCP response curve [14] can be approximated for small end-to-end drop rates by

$$cwnd_r \approx \sqrt{\frac{1.5}{P_r}} \tag{8}$$

[2]This can be derived by considering the congestion window size at equilibrium through a differential equation model of *cwnd* or the expectation of a stochastic model of *cwnd*.

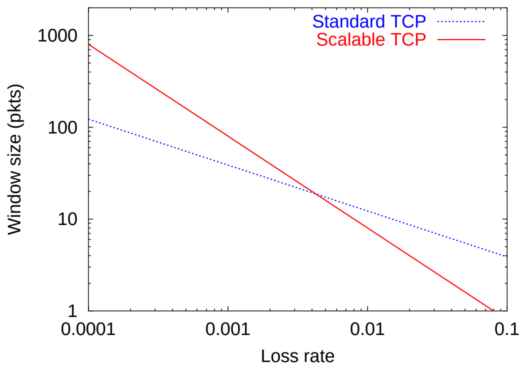

Figure 3: Response curves for traditional TCP and Scalable TCP.

The two response curves have different forms for the multiplicative function of P_r; the two schemes cannot have average windows of the same value for all end-to-end loss rates P_r. However, all that is needed is a suitable evolutionary approach that allows connections to better use bandwidth in wide area networks when it is available. The argument that follows was first introduced in [6]. Traditional TCP connections can not effectively use large windows and in practice have a limited amount of socket receive and send buffer memory available, so they will tend not to have a windows greater than a certain size; call this the *legacy window* size $lwnd$. Associate with this window size the *legacy loss rate*, P_l, which is the maximum packet loss rate needed to support windows larger than $lwnd$. Suppose Scalable TCP uses the traditional congestion window update algorithm when $cwnd \leq lwnd$ and the Scalable TCP congestion window update algorithm for $cwnd > lwnd$. The sharing properties of Scalable TCP can then be considered in two states. For levels of congestion with drop rates higher than p_l the Scalable TCP connections use the traditional TCP algorithm and receive the same share as a traditional TCP stack.[3] For levels of congestion with drop rates less than p_l legacy connections will have a window of at least $lwnd$. Scalable TCP connections will receive larger windows than legacy connections but the legacy connections are never starved of bandwidth.

The choice of the value $lwnd$ is a policy decision. If $lwnd = 16$, it is only when traditional TCP connections have a window of about 420 that Scalable TCP connections of the same round trip time will receive twice the bandwidth. This suggests that concerns about Scalable TCP receiving a higher bandwidth than traditional TCP connections with windows greater than $lwnd$ should not arise until the window size is already large enough for there to be concerns about TCP packet loss recovery times. For the purposes of this paper we will assume that $lwnd$ is 16 packets; this corresponds to 24KB with 1500 byte segments and a legacy loss rate, p_l, of 5.86×10^{-3}. The response curves for traditional TCP and Scalable TCP are plotted for an $lwnd$ of 16 in Figure 3.

To ensure a continuous and decreasing response curve, the Scalable TCP response curve must pass through the point $(p_l, lwnd)$ giving the following constraint on a and b

$$\frac{a}{b} = p_l * lwnd \approx \sqrt{1.5 p_l} \tag{9}$$

The number of free variables is now reduced to one; choosing b fixes a.

[3]There is not an intrinsic problem with using the Scalable TCP algorithm in a small window regime; previous studies [11, 13] suggest that there may be benefits to doing so in the context of ECN IP networks. However Scalable TCP connections would receive a smaller share of the bandwidth, would react more slowly to congestion, and may alter the dynamics of existing traffic. These effects could make evolution through incremental deployment more difficult and so are avoided in the design presented here.

3.2 Instantaneous rate variation

The instantaneous rate of a TCP connection probes around a mean value giving it a share of the available capacity. The size of this stochastic rate variation for the Scalable TCP congestion window update algorithm has been studied previously [13].[4] The coefficient of variance for the instantaneous sending rate is

$$CoV\left(x_r\right) = CoV\left(\frac{cwnd_r}{T_r}\right) \sim \sqrt{\frac{b}{2}} \tag{10}$$

provided $P_r \downarrow 0$. This suggests that b should be chosen as small as possible to reduce instantaneous rate variation, a conclusion that agrees with intuitive arguments based on the packet loss recovery times shown in Figure 2. It appears sensible not to make the algorithm have a rate variation larger than traditional TCP, so b should satisfy $b \leq \frac{1}{2}$.

The Scalable TCP algorithm responds to congestion events at most once per round trip time. Therefore it is necessary that the window expansion and contraction cycle lasts longer than a round trip time.[5] Using the packet loss recovery time of Scalable TCP and noting that b is the only free variable, this constraint becomes

$$b > 1 - \sqrt{\frac{1}{1.5p_l}} \tag{11}$$

This constraint is often trivially satisfied; with a *lwnd* of 16, it becomes satisfied for any $b > 0$ because $1 - \sqrt{\frac{1}{1.5p_l}} \approx -113 < 0$.

3.3 Convergence

Convergence speed is of significance to an elastic rate protocol that must adapt to changing network conditions on reasonable timescales. Ideally convergence should happen instantaneously. However the use of packet loss as a signaling channel, the need to provide compatibility with legacy traffic and to use minimal cost network devices, can make this goal difficult. Suppose that at time t_0 a sudden overload shock occurs and P_r increases. Then a source will reduce its sending rate upon receiving feedback by a factor of $\frac{1}{2}$ in less than

$$\frac{log(\frac{1}{2})T_r}{log(1-b)} \tag{12}$$

In fact this is an overestimate of the time needed. Any overload that causes loss and delay will lead to a lower sending rate because acknowledgments from the receiver are needed to release packets into the network; this *self-clocking* is a robust mechanism that reacts within a round trip time to overload events. Traditional TCP congestion control corresponds to a choice of $b = \frac{1}{2}$; a fairly rapid convergence speed in the face of overload.

In response to a sudden increase in the available capacity on a route, $P_r \downarrow 0$, and the time taken for the source to increase its sending rate by a factor of 2 is

$$\frac{log(2)T_r}{log(1+a)} \tag{13}$$

By contrast a traditional TCP connection would require $cwnd_r(t_0)$ round trip times to respond to the increase in available capacity. The Scalable TCP algorithm responds more effectively to changes in available capacity when window sizes are large.

These convergence properties suggest that b (and also a)[6] should be chosen as large as possible for fastest convergence. This conflicts with the desire to keep the instantaneous rate variation small which requires b to be small, from Equation 10.

[4]The responses were considered in the context of an ECN implementation. However these results provide a good approximation with large windows, low drop probabilities, and the constraint in Equation 11.

[5]ECN implementations reacting to each congestion notification do not necessarily suffer from this limitation.

[6]a is proportional to b by Equation 9; so a large b gives a large a.

b	a	Coefficient of variation for rate	Packet loss recovery time	Time to halve rate	Time to double rate
$\frac{1}{2}$	$\frac{2}{50}$	0.50	$17.7T_r$ or 3.54s	T_r or 0.20s	$17.7T_r$ or 3.54s
$\frac{1}{4}$	$\frac{1}{50}$	0.35	$14.5T_r$ or 2.91s	$2.41T_r$ or 0.48s	$35T_r$ or 7.00s
$\frac{1}{8}$	$\frac{1}{100}$	0.25	$13.4T_r$ or 2.68s	$5.19T_r$ or 1.04s	$69.7T_r$ or 13.9s
$\frac{1}{16}$	$\frac{1}{200}$	0.18	$12.9T_r$ or 2.59s	$10.7T_r$ or 2.15s	$139T_r$ or 27.8s

Table 2: Properties of a Scalable TCP connection with a variety of parameter settings for a general round trip time or at 200ms.

Table 2 shows the properties of a Scalable TCP connection for a general round trip time and when it is equal to 200ms; these choices of a and b are compatible with a legacy window of 16 packets. The setting of a and b is a policy choice determined by which system properties are deemed to be most important. It would appear that the variability of choosing $b = \frac{1}{2}$ is too large. However the slow convergence times of $b = \frac{1}{16}$ would suggest that choosing b between $\frac{1}{4}$ and $\frac{1}{8}$ is desirable. In this paper $b = \frac{1}{8}$ is selected because it offers a good balance between rate fluctuation and convergence time. The choosing of the optimal parameter in this range appears to make only a marginal difference to the theoretical dynamics of the algorithm; further experimentation using more implementations and real workloads will help to refine this choice.

3.4 Stability

It has been shown [16] that for heterogeneous round trip times and arbitrary network topologies, the generalised Scalable TCP algorithm is locally stable[7] about its equilibrium provided

$$a < \frac{p_j(\hat{y}_j)}{\hat{y}_j p'_j(\hat{y}_j)} \qquad \forall j \in J \tag{14}$$

where $\hat{y}_j$ is the equilibrium rate at each link, $p_j(y)$ is the probability of loss at link j for an arrival rate y, and J is the set of all links.

For example, assuming Poisson packet arrivals,[8] the scheme is stable if FIFO network buffers are provisioned to be of size at most $\frac{1}{a}$. Hence if the network buffers can be configured the system can be made stable in a control theoretic sense. A control theoretic approach to the design of a stable and scalable TCP using ECN is given in [11]. Further improvements and enhancements are possible with the use of adaptive queue management (AQM) schemes at network devices but is beyond the scope of this paper.

4 Experiments

Scalable TCP was implemented in the network stack of the Linux 2.4.19 operating system. This kernel implements a sophisticated TCP stack supporting the following relevant standards: TCP extensions for high performance[9] [10], SACK [7], and D-SACK [8]. The stack also implements packet retransmission timeout checking to detect lost packets[10], reordering detection using D-SACK, rate halving, and burst limiting. The Scalable TCP patch[11] adds the congestion window algorithm changes, scalings to kernel buffers, the removal of special case small packet handling in

[7]This is in the sense that the differential equations for all the sending rates are locally stable with respect to the feedback loop controlling them.

[8]Other traffic models can also be considered and the results are qualitatively similar; see [11] for some examples.

[9]This provides the following enhancements: window scaling, timestamping, and protection against wrapped sequence numbers.

[10]This is similar to that used in TCP Vegas [2] to quickly detect losses with duplicate acknowledgements.

[11]The Scalable TCP patch used for the experiments in this paper can be downloaded from: `http://www-lce.eng.cam.ac.uk/~ctk21/scalable/`.

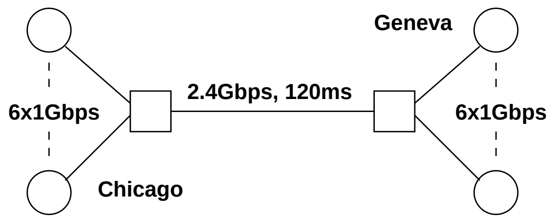

Figure 4: Testbed topology used for experiments.

Number of flows	2.4.19 TCP	2.4.19 TCP with gigabit kernel modifications	Scalable TCP
1	7	16	44
2	14	39	93
4	27	60	135
8	47	86	140
16	66	106	142

Table 3: Number of 2 Gigabyte transfers completed in 1200 seconds.

the SysKonnect driver, and debug counters. The scaling of kernel buffers increases the send and receive queues that lie between the kernel and device driver. This is needed because scheduling timeslices have remained constant while interface speeds have increased.[12] The SysKonnect device driver for Linux 2.4.19 copies small packets into their own buffer to conserve memory. In order to optimize for speed rather than space efficiency, the driver's interrupt handling routine was changed to not make this extra copy. Both of these changes were simple and significantly improved TCP throughput; they will be termed the gigabit kernel modifications. In order to adjust for the effect of delayed acknowledgments a was set to 0.02. The implementation of byte counting [1], which updates the congestion window in proportion to the exact number of bytes acknowledged, would remove the need to adjust for delayed acknowledgments.

The DataTAG testbed consists of 12 high performance PCs that have Supermicro P4DP8-G2 motherboards with dual 2.4GHz Xeon processors and 2 gigabytes of memory. SysKonnect SK-9843 Gigabit Ethernet cards on a 133MHz/64bit PCI bus provided connectivity to the testbed network. 6 servers are located at CERN, Geneva, and 6 servers at StarLight, Chicago. The clusters are connected through two Cisco 76xx routers with a 2.4Gbps packet over SONET link between Geneva and Chicago. The PCs are connected to each routers through gigabit Ethernet ports. This topology is shown in Figure 4. The round trip time for a ping from Geneva to Chicago was 120ms. In the experiments that follow the interface between Geneva and Chicago had a FIFO queue of 2048 packets. All the other gigabit Ethernet interfaces on the routers had the factory default setting of a 40 packet FIFO queue. At most 9% of the bandwidth-delay product is available as buffers on the path; this trend towards a decrease in available buffering delay is likely to continue due to the cost of implementing highspeed memory systems in network devices.

Three sender side test cases are compared: TCP in an unaltered Linux 2.4.19 kernel, TCP in a Linux 2.4.19 kernel with the gigabit kernel modifications, and Scalable TCP in a Linux 2.4.19 kernel with the gigabit kernel modifications. The receivers used an unaltered Linux 2.4.19 kernel in all cases. The experiments were designed to explore the performance of Scalable TCP for bulk data transfer as could be found in wide area scientific networks.

[12]The transmit side queue is limited by a device's *txqueue* variable. The receive side queue size is set by the sysctl variable *net.core.netdev_max_backlog*. These were increased to 2000 and 3000 respectively to hold the increased number of packets that can arrive during a period where the operating system cannot process them immediately.

Component	Probability density function	Parameters	Mean
Think times (sec)	(Pareto) $p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$, $x > k$	$k = 10.0, \alpha = 2.0$	20.0
Objects per page	(Pareto) $p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$, $x > k$	$k = 3.0, \alpha = 1.5$	9.0
Request file sizes (bytes)	(Pareto) $p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$, $x > k$	$k = 12000, \alpha = 1.2$	72000
Inter object times (sec)	(Pareto) $p(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$, $x > k$	$k = 0.5, \alpha = 1.5$	1.5

Table 4: Summary of distributions and parameters used in the Web user TCP connection model.

Type of bulk transfer users	Web traffic transferred	2 Gigabyte transfers completed
No bulk transfers	65GB	n/a
TCP in 2.4.19	65GB	36
TCP in 2.4.19 with buffer scaling	65GB	58
Scalable TCP	65GB	96

Table 5: Performance with 4200 concurrent Web users and 8 bulk transfer users over 1200 seconds.

4.1 Basic performance

In these tests 4 server and receiver pairs were used with TCP flows distributed evenly across the 4 machines. Each receiver in Chicago would requested a file of size 2 Gigabytes from its associated server in Geneva. The server responded by transferring 2 Gigabytes of data (from memory) back to the receiver in Chicago. Upon completion of the 2 Gigabyte transfer the connection was completed and another request was initiated. This was intended to capture some slow-start and termination dynamics. In all cases each TCP socket had send and receive buffers set to 64MB; this allowed a single flow to make full use of any bandwith available to it.

Table 3 shows the results of these experiments. A significant throughput improvement of 60% to 180% was observed simply by scaling the internal Linux kernel buffers and removing the copying of small packets in the receive path of the SysKonnect device driver. The Scalable TCP congestion control algorithm further increased throughput by 34% to 175% over that observed with traditional TCP using the gigabit kernel modifications. Using 16 Scalable TCP flows across four machines acheived 81% of the maximal performance possible over a saturated 2.4Gbps link after accounting for the required IP and TCP header overhead incurred with packets of size 1500 bytes.[13] The Linux 2.4.19 kernel with gigabit kernel modifications could get 61% of the maximal 2.4.Gbps performance with 16 flows. A standard Linux 2.4.19 kernel achieved at most 38% of the maximal performance with 16 flows.

4.2 Performance with Web traffic

These tests attempted to measure the impact on Web traffic of large bulk transfer users. In particular they assessed whether Scalable TCP has a detrimental effect on existing TCP users. In these tests, three receiver and server pairs each generated traffic equivalent to 1400 active Web users.[14] Two machine pairs generated transfer requests of 2 Gigabytes in size, in the same way as the basic throughput test, with eight transfers in progress across the two machines at any one time. The parameters used for the Web traffic model are given in Table 4; these parameters are the same as those measured in [5] to generate self-similar traffic. The Web traffic was made repeatable in the sense that the sample paths of user think times, embedded pages, inter-object times and

[13]For a combined IP and TCP header using the timestamp option the maximal capacity is about 96% of the stated interface capacity.

[14]This traffic is not completely representative of Web traffic observed in real networks because only one round trip time was available for experimental purposes.

page-sizes were the same for a given user across each test. This repeatability allowed the Web traffic to be run in isolation and then with additional traffic to measure the impact of the bulk traffic on the Web transfers. Table 5 displays the results of the experiments on mixing the traffic types.

In none of the tests did the Web traffic experience any noticeable change in throughput. This offers evidence to suggest that the design of Scalable TCP has indeed provided a solution with negligible impact on existing traffic. The standard Linux 2.4.19 kernel with no modifications achieved 40% of the maximal possible system throughput over the time period. Applying the gigabit kernel modifications improved traditional TCP performance and achieved 52% of the maximal possible throughput. The bulk transfers using the Scalable TCP algorithm boosted the total traffic transferred to 75% of the maximum possible throughput.

5 Related work

Several authors have made the case for using TCP Vegas [2, 17] and similar variants [3] in highspeed networks. The argument proceeds by observing that TCP Vegas uses network buffer delay as an implicit congestion signal as opposed to drops. Hence if network buffer delay can be controlled and used as a signaling mechanism, it should be possible to run the network at very high utilizations. This approach may prove to be successful but is challenging to implement. To succeed TCP Vegas implementations are needed that can run robustly in environments where noise affects delay estimates; noise could arise from heterogeneous network buffering schemes, operating system scheduling, network firewall processing, and cross traffic which does not control buffer delay such as traditional TCP or UDP streams.

Others have used mechanisms to make one logical connection behave like multiple TCP connections to improve performance in high bandwidth wide area networks; this can be acheived either at the transport layer [4] or at the application layer by opening multiple connections. The results displayed in Table 3 show that this can be a pragmatic solution to improve throughput. However it can be difficult to tune in a way that consistently provides good performance without causing a detrimental effect on existing network traffic when congestion occurs.

This work builds on the Highspeed TCP proposal [6] and uses the same arguments to achieve good sharing with legacy applications. Scalable TCP is simpler to implement than the parameterized Highspeed TCP algorithm due to its use of constants in the window update algorithm. The work also shares the analysis and design methods used to engineer other ECN TCP variants [11, 13].

6 Conclusion

Scalable TCP presents a simple change to the congestion window update algorithm which improves throughput in highspeed wide area networks. The performance improvement can be dramatic for senders using the Scalable TCP algorithm in bulk transfer networks; the improvement attributable to the algorithm can sometimes be over 100%. The scheme also promises to interoperate well with legacy traffic; results from the experiments conducted with Web traffic using traditional TCP stacks in parallel with several Scalable TCP flows performing bulk transfers showed negligible impact on the Web traffic transferred. A surprising result of the experiments performed is that simple optimizations to kernel device drivers can improve traditional TCP performance by over 100% when compared to a standard kernel.

Future work is needed to consider the impact of heterogeneous round trip times. There may be a requirement to correct the bias TCP has towards connections with smaller round trip times; the methods used for scalable ECN variants [11] could provide a good starting point for such modifications. Additional work could also consider more complex workload models which capture the needs of the applications that may be run on highspeed wide area networks.

7 Acknowledgments

Jean-Philippe Martin-Flatin offered valuable comments on early drafts of this paper and useful coding advice. Helpful discussions on the design of Scalable TCP were had with Glenn Vinnicombe, Sally Floyd, and Frank Kelly. Thanks also go to the DataTAG testbed support teams at CERN, Caltech and StarLight. This work was funded by the IST Programme of the European Union (grant IST-2001-32459, DataTAG project), the Royal Commission for the Exhibition of 1851, and AT&T Labs - Research.

References

[1] M. Allman. TCP Byte Counting Refinements. *ACM Computer Communication Review*, 29(3), July 1999.

[2] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[3] D. H. Choe and S. H. Low. Stabilized Vegas. In *Proc. of the 39th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2002.

[4] J. Crowcroft and P. Oechslin. Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP. *Computer Communication Review*, 28(3), July 1998.

[5] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control. In *SIGCOMM 1999*, Boston, MA, August 1999.

[6] S. Floyd. HighSpeed TCP for Large Congestion Windows. *Internet Draft <draft-floyd-tcp-highspeed-01.txt>*, August 2002. Work in progress.

[7] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. *Internet RFC 2883*, July 2000.

[8] S. Floyd, K. K. Ramakrishnan, and D. Black. The addition of explicit congestion notification (ECN) to IP. *Internet RFC 3168*, September 2001.

[9] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM 1988*. An updated version is available via `ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z`.

[10] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High performance. *Internet RFC 1323*, May 1992.

[11] T. Kelly. On Engineering a Stable and Scalable TCP Variant. Technical Report CUED/F-INFENG/TR.435, Laboratory for Communication Engineering, Cambridge University, June 2002.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. *Internet RFC 2018*, October 1996.

[13] A. Misra and T. J. Ott. Performance Sensitivity and Fairness of ECN-Aware 'Modified TCP'. In *Networking 2002: Networking Technologies, Services, and Protocols; Performance of Computer and Cmmmunication Networks; and Mobile and Wireless Communications, Second International IFIP-TC6 Networking Conference Proceedings.*

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[15] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley, 1994.

[16] G. Vinnicombe. On the stability of networks operating TCP-like congestion control. In *Proc. of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[17] E. Weigle and W. Feng. A Case for TCP Vegas in High-Performance Computational Grids. In *Proc. of the 9th IEEE International Symposium on High performance Distributed Computing (HPDC'01)*, San Francisco, CA, August 2001.