The TCP Bandwidth-Delay Product revisited: network buffering, cross traffic, and socket buffer auto-sizing

Manish Jain, Ravi S. Prasad, Constantinos Dovrolis
Networking and Telecommunications Group
College of Computing
Georgia Tech
{jain,ravi,dovrolis}@cc.gatech.edu

Abstract— *TCP is often blamed that it cannot use efficiently network paths with high Bandwidth-Delay Product (BDP). The BDP is of fundamental importance because it determines the required socket buffer size for maximum throughput. In this paper, we re-examine the BDP concept, considering the effects of network buffering and cross traffic on the 'bandwidth' and 'delay' characteristics of a path. We show that, with careful socket buffer sizing, a bulk TCP transfer can saturate a network path independent of the BDP or the available network buffers. In a non-congested path, there is a certain socket buffer size (which depends on the cross traffic type) that maximizes the throughput of a bulk TCP transfer. In a congested path, the TCP throughput is maximized when the connection is limited by the congestion window, rather than by the socket buffers. Finally, we present an application-layer mechanism (SOBAS) that automatically adjusts the socket buffer size close to its optimal value, based on direct measurements of the maximum received throughput and of the round-trip time, without requiring prior knowledge of the path characteristics.*

Keywords: TCP throughput, router buffers, available bandwidth, bottleneck bandwidth, round-trip time, fast long-distance networks.

I. Introduction

There is a significant interest recently in end-to-end performance over high-bandwidth and long-distance networks [1]. In particular, it is the scientific community that pushes the edge of network performance with applications such as distributed simulation, remote colaboratories, and with huge transfers (gigabytes or more). Typically, such applications run over well-provisioned networks (Internet2, ESnet, GEANT, etc) built with high-bandwidth links (OC-12 or higher) that are lightly loaded in most of the time. Additionally, through the gradual deployment of Gigabit Ethernet interfaces, congestion also becomes rare at the network edges and end-hosts. With all this network bandwidth, it is not surprising that users expect a superb end-to-end performance. However, this is not always the case. A recent measurement study at the Internet2 showed that 90% of the 'bulk' TCP transfers (i.e., transfers of more than 10MB) typically receive less than 5Mbps [2]. This is an unsatisfactory result, given that Internet2 uses OC-48 links that are rarely utilized by more than 30%, while most universities connected to Internet2 also operate high-bandwidth and well-provisioned networks.

A popular belief is that a major reason for the relatively low end-to-end throughput in fast long-distance networks is TCP. This is either due to TCP itself (e.g., its congestion control algorithms and parameters), or because of local system configuration (e.g., default TCP socket buffer size) [3], [4].[1] TCP is blamed that it is slow in capturing the available bandwidth of high-performance networks, especially in transfers over long-distance paths. Two issues commonly identified as the underlying reasons are:
1. Limited socket buffers at the TCP sender or receiver impose a conservative upper bound on the effective window of the transfer, and thus on the maximum achievable throughput.
2. Packet losses cause a large and multiplicative window reduction, and a subsequent slow (linear) window increase rate, causing an overall low average throughput.
Other TCP-related issues that often impede performance are multiple packet losses at the end of slow-start (commonly resulting in timeouts), the inability to distinguish between congestion and random packet losses, the use of small segments, the coarse granularity of the retransmission timeout, or the initial value of the *ssthresh* parameter [5], [6].

Networking research has focused on these problems, pursuing mostly three approaches: modified congestion control algorithms [7], [8], [9], [10], parallel TCP transfers [11], [12], [13], [14], and improved socket buffer sizing [5], [15], [16], [17]. Modified congestion control schemes, possibly

This work was supported by the 'Scientific Discovery through Advanced Computing' program of the US Department of Energy (award number: DE-FC02-01ER25467), and by the 'Strategic Technologies for the Internet' program of the US National Science Foundation (award number: 0230841). Please do not distribute without the authors' permission.

[1] Other reasons include Fast Ethernet duplex mismatches, lossy cables, under-buffered switches/routers, and poorly written applications.

with cooperation from routers [9], can lead to significant benefits for both applications and networks, and they can also address other important issues, such as the detection of congestion unresponsive traffic. Modifying TCP, however, is an enormously difficult task today, given the millions of existing TCP installations and the fairness issues that would arise during deployment of diverse congestion control algorithms. Parallel TCP connections can increase the aggregate throughput that an application receives. They also raise fairness issues, however, because an aggregate of N connections decreases its aggregate window by a factor $\frac{1}{2N}$, rather than $\frac{1}{2}$, upon a packet loss. Also, the aggregate window increase rate is N times faster than that of a single connection. This paper follows the socket buffer sizing approach, rather than modifying TCP or using parallel connections. Socket buffer sizing can be performed by applications, and so it does not require changes at the TCP implementation or protocol. Also, it is a mechanism that can work complementary with parallel connections. Even if TCP evolves to a different protocol in the future, we believe that it is still important to consider how we can improve application performance in the shorter term, using the existing TCP incarnation.

How can an application determine an appropriate size for the socket buffer at the sender and receiver of a TCP transfer? A first constraint is that the socket buffers cannot exceed the memory that the operating system makes available for that connection. Throughout the paper, we assume that the end-hosts have enough memory and that this constraint is met. This is widely the case today, with the exception perhaps of busy Web or file servers. A second constraint is that the two socket buffers, or actually the smaller of the two, should be sufficiently large so that the transfer can saturate the underlying network path. This leads us to a fundamental concept in any window-controlled transport protocol: the *Bandwidth-Delay Product (BDP)*. Specifically, suppose that the bottleneck link of a path has a transmission capacity ('bandwidth') of C bps and the path between the sender and the receiver has a Round-Trip Time (RTT) of T sec. The connection will be able to saturate the path, achieving the maximum possible throughput C, if its effective window is $C \times T$. This product is historically referred to as BDP. For the effective window to be $C \times T$, however, the smaller of the two socket buffers should be equally large. If the size of that socket buffer is less than $C \times T$, the connection will underutilize the path. If it is more than $C \times T$, the connection will overload the path, and depending on the amount of network buffering, it will cause congestion, packet losses, window reductions, and possibly throughput drops.

The previous interpretation of the BDP, and its relation to TCP throughput and socket buffer sizing, are well-known in the networking literature. As we argue in Section II, however, the socket buffer size should be equal to the BDP *only in the case of a path that does not carry competing traffic ('cross traffic') and that does not introduce queueing delays*. The presence of cross traffic means that the 'bandwidth' of a path will not be C, but somewhat less than that. Also, packet buffers in the network routers can cause queueing delays, meaning that the RTT of the path will be larger than the fixed delay T. Given these two additional effects (cross traffic load, and queueing delays), how should we define the BDP of a network path? Should we interpret the 'bandwidth' term of the BDP as the capacity C, the *available bandwidth* which remains unclaimed from cross traffic, the 'fair share' in the path (for some definition of fairness), or as something different than the above? And how should we interpret the 'delay' term of the BDP? Is it the minimum possible RTT, the average RTT *before* the TCP connection starts (including queueing delays due to cross traffic), or the average RTT *after* the TCP connection starts? In Section III, we review the previous work in the area of socket buffer sizing, arguing that the BDP has been given different interpretations in the past and that it is still unclear what is the socket buffer size that maximizes the throughput of a bulk TCP transfer.

The first objective of this paper is to *examine the effects of the socket buffer size, the amount of network buffering, and the cross traffic type on the throughput of a bulk TCP transfer*. In Section IV, we focus on the effect of network buffering. We show that, with appropriate socket buffer sizing, a TCP transfer can saturate a network path, independent of how large the BDP is, and independent of the available network buffering. In Sections V and VI, we examine the effects of cross traffic on the TCP throughput. The type of cross traffic is crucial, as it determines whether and by how much the latter will decrease its send-rate upon the initiation of a new TCP transfer. We identify four distinct common types of cross traffic: rate-controlled UDP flows, TCP transfers limited by congestion, TCP transfers limited by socket buffers, and TCP transfers limited by size. We show that, depending on the cross traffic type, the Maximum Feasible Throughput of a TCP connection can be the Available Bandwidth [18], the Bulk Transfer Capacity (BTC) [19], or the maximum TCP throughput that does not cause packet losses in the path. An important conclusion from Sections V and VI is that, *independent of cross traffic and network buffering, there is a certain socket buffer size that maximizes the throughput of a bulk TCP transfer*.

The second objective of this paper is to *develop an application-layer mechanism that can automatically set the socket buffer size so that the TCP transfer receives its Maximum Feasible Throughput, or at least close to that*. In Section VII, we describe the proposed *SOcket Buffer Auto-Sizing (SOBAS)* mechanism. SOBAS estimates the path's BDP from the maximum TCP goodput measured by the receiving application, as well as from out-of-band RTT measurements. We emphasize that *SOBAS does not require changes in TCP*, and that it can be integrated in principle with any TCP-based bulk data transfer application. SOBAS has been evaluated through simulations and Internet experiments. Simulations are used to compare SOBAS with the Maximum Feasible Throughput and with other socket buffer sizing techniques under the same traffic conditions. Experimental results provide us with confidence that SOBAS works well, even with all the additional measurement inaccuracies, traffic dynamics, non-stationarities, and noise sources of a real Internet path. A key point about SOBAS is that it does not require prior knowledge or estimation of path characteristics such as the end-to-end capacity or available bandwidth. We conclude in Section VIII.

II. What does Bandwidth-Delay Product mean?

Consider a unidirectional TCP transfer from a sender $\mathcal{SND}$ to a receiver $\mathcal{RCV}$. TCP is window-controlled, meaning that $\mathcal{SND}$ is allowed to have up to a certain number of transmitted but unacknowledged bytes, referred to as the *send-window* W_s, at any time. The send-window is limited by

$$W_s = \min\{W_c, W_r, B_s\} \tag{1}$$

where W_c is the sender's *congestion window* [20], W_r is the *receive-window* advertised by $\mathcal{RCV}$, and B_s is the size of the *send-socket buffer* at $\mathcal{SND}$. The receive-window W_r is the amount of available *receive-socket buffer* memory at $\mathcal{RCV}$, and is limited by the receive-socket buffer size B_r, i.e., $W_r \leq B_r$. So, the send-window of a transfer is limited by:

$$W_s = \min\{W_c, S\} \tag{2}$$

where $S = \min\{B_s, B_r\}$ *is the minimum of the two socket buffer sizes*. If the send-window is limited by W_c, we say that the transfer is *congestion-limited*; otherwise, if the send-window is limited by S, we say that the transfer is *buffer-limited*. A transfer can be congestion-limited or buffer-limited at different time periods. If $T(W_s)$ is the connection's RTT when the send-window is W_s, the *transfer's throughput* is

$$R = \frac{W_s}{T(W_s)} = \frac{\min\{W_c, S\}}{T(W_s)} \tag{3}$$

Note that the RTT can vary with W_s because of queueing delays due to the transfer itself.

We next describe a model for the network path $\mathcal{P}$ that the TCP transfer goes through. The bulk TCP transfer that we focus on is referred to as the *target transfer*; the rest of the traffic in the path is referred to as *cross traffic*. The *forward path* from $\mathcal{SND}$ to $\mathcal{RCV}$, and the *reverse path* from $\mathcal{RCV}$ to $\mathcal{SND}$, are assumed to be fixed and unique for the duration of the target transfer. Each link i of the forward/reverse path transmits packets with a *capacity* of C_i bps, causes a *fixed delay* τ_i, and it has *link buffers* that can store B_i packets. Arriving packets are discarded in a Drop-Tail manner when the corresponding buffer is full. The *minimum RTT* of the path is the sum of all fixed delays along the path $T_m = \sum_{i \in \mathcal{P}} \tau_i$.

Also, let ρ_i be the initial *average utilization* of link i, i.e., the utilization at link i *prior* to the target transfer. The *available bandwidth* A_i of link i is then defined as $A_i = C_i \times (1 - \rho_i)$. Due to cross traffic, link i introduces queueing delays together with the fixed delay τ_i. Let $\bar{\tau}_i \geq \tau_i$ be the *average delay* at link i, considering both fixed and queueing delays, prior to the target transfer. The *exogenous RTT* of the path is the sum of all average delays along the path $T_e = \sum_{i \in \mathcal{P}} \bar{\tau}_i$ before the target transfer starts.

Adopting the terminology of [18], we refer to the link of the forward path $\mathcal{P}_f$ with the minimum capacity $C = \min_{\mathcal{P}_f}\{C_i\}$ as the *narrow link*, and to the link with the minimum available bandwidth $A = \min_{\mathcal{P}_f}\{A_i\}$ as the *tight link*. We say that a link is *saturated* when its utilization is 100% (i.e., A_i=0), and that it is *congested* when it drops packets due to buffer overflow (i.e., non-zero loss rate). Note that a link can be saturated but not congested, or congested but not saturated. We assume that the only saturated and/or congested link in the forward path is the tight link. *A path is called congested or saturated, if its tight link is congested or saturated, respectively*.

The narrow link limits the maximum throughput C that the target transfer can get. The tight link, on the other hand, is the link that becomes saturated and/or congested after the target transfer starts, when the latter has a sufficiently large send-window. The buffer size of the tight link is denoted by B_t, while $\tilde{B}_t \leq B_t$ is the *average available buffer size* at the tight link prior to the target transfer. We refer to C, T_m, and B_t as the *structural path characteristics*, and to A, T_e, and $\tilde{B}_t$ as the *dynamic path characteristics* prior to the target transfer. The dynamic path characteristics depend on the cross traffic.

Equation (3) shows that the throughput of the target transfer depends on the minimum socket buffer size S. If we view the throughput R as a function $R(S)$, an important question is: *given a network path $\mathcal{P}$, with certain structural and dynamic characteristics, for what value(s) of the socket buffer size $\hat{S}$ the target transfer throughput $R(S)$ is maximized?* We refer to the maximum value of the function $R(S)$ as the *Maximum Feasible Throughput (MFT)* $\hat{R}$. The conventional wisdom, as expressed in networking textbooks [21], operational handouts [4], and research papers [16], is that the socket buffer size S should be equal to the Bandwidth-Delay Product of the path, where 'bandwidth' is the capacity of the path C, and 'delay' is the minimum RTT of the path T_m, i.e., $S = C \times T_m$. Indeed, if the send-window is $W_s = C \times T_m$, and *assuming that there is no cross traffic in the path*, the tight link becomes saturated but not congested, and the target transfer achieves its Maximum Feasible Throughput $\hat{R} = C$. Thus, in the case of no cross traffic, $\hat{S} = C \times T_m$.

In practice, a network path always carries some cross traffic, and thus $A < C$. If the socket buffer size is set to $C \times T_m$, the target transfer will saturate the tight link, and depending on the number of buffers at the tight link B_t, it may also cause congestion. Congestion, however, causes multiplicative drops in the target transfer's send-window, and, potentially, throughput reductions as well. Thus, the amount of buffering B_t at the tight link is an important factor for socket buffer sizing, as it determines the point at which the tight link becomes congested. The presence of cross traffic has two additional implications. First, cross traffic causes queueing delays in the path, and thus the initial RTT of the target transfer becomes the exogenous RTT T_e, rather than the minimum RTT T_m. With larger B_t and with burstier cross traffic, the difference between T_m and T_e becomes more significant. Second, if the cross traffic is TCP (or TCP-friendly), it will react to the initiation of the target TCP transfer reducing its throughput, either because of packet losses, or because the target transfer has increased the RTT in the path. Thus, the target transfer can achieve a higher throughput than what was the initial available bandwidth A.

In the next section, we review the previous work in the area of socket buffer sizing, and identify several interpretations that have been given to the BDP:

1. **BDP$_1$:** $S = C \times T_m$
2. **BDP$_2$:** $S = C \times T_e$
3. **BDP$_3$:** $S = A \times T_m$
4. **BDP$_4$:** $S = A \times T_e$
5. **BDP$_5$:** $S = \text{BTC} \times T_a$
6. **BDP$_6$:** $S = S_\infty$ (where $S_\infty > \max W_c$)

The first four BDP definitions should be clear from the previous model. The bandwidth term in BDP$_5$ is the *Bulk Transfer Capacity (BTC)*, i.e., *the average throughput of a bulk congestion-limited TCP transfer* [22]. It can be argued that the BTC is the 'fair-share' of the target transfer in the path, according to TCP's bandwidth sharing properties. T_a *is the average RTT of the path, after the target transfer has started*, and so it includes the queueing load due to the target transfer. BTC is determined by the congestion window and the average RTT of the target transfer, and so BDP$_5$ is related to the *average congestion window* of the transfer.

In BDP$_6$, S is set to a sufficiently large value, so that it is always larger than the congestion window. So, the meaning of BDP$_6$ is that connection should be always congestion-limited. Of course, in the absence of any previous information about the path, it is hard to know how large the maximum congestion window will be. Note that BDP$_5$ and BDP$_6$ are different: in the former, the connection may be buffer-limited during parts of its lifetime, while in the latter the connection's send-window is never limited by the socket buffers.

III. Previous work on socket buffer sizing

There are several measurement and estimation techniques for bandwidth-related metrics, such as the capacity, available bandwidth, or Bulk Transfer Capacity of a path [18], [22], [23], [24]. An application of such techniques is that they can be used to estimate the bandwidth term of a path's BDP.

An auto-tuning technique that is based on active bandwidth estimation is the *Work Around Daemon (WAD)* [5]. WAD uses *ping* to measure the minimum RTT T_m prior to the start of a TCP connection, and *pipechar* to estimate the capacity C of the path [25]. In other words, [5] attempts to set the socket buffer size as in BDP$_1$. A similar approach is taken by the NLANR Auto-Tuning FTP implementation [26]. In that work, however, the socket buffer sizing is based on the median, rather than the minimum, of the bandwidth-delay product measurements and so it is closer to BDP$_2$. Similar BDP interpretations are given at the manual socket buffer sizing guidelines of [4] and [15].

A different bandwidth metric is considered in [27]. That work proposes a TCP variant in which the send-window is adjusted based on the available bandwidth of a path. The proposed protocol is called TCP-Low Priority (TCP-LP), and it is able to utilize any excess network bandwidth that would not be used normally by the regular TCP workload [27]. Even though TCP-LP is not a socket buffer sizing scheme, it relates more to BDP$_3$ or BDP$_4$ because it is the available bandwidth, rather than the capacity, that limits the transfer's send-window.

The first proposal for automatic TCP buffer tuning was [16]. The goal of that work was to allow a host (typically a server) to fairly share kernel memory between multiple ongoing connections. The proposed mechanism, even though simple to implement, requires changes in the operating system. An important point about [16] is that the BDP of a path was estimated based on the congestion window (*cwnd*) of the TCP connection. Thus, the socket buffer sizing objective in that work was similar to BDP$_5$. The receive-socket buffer size was set to a sufficiently large value so that it does not limit the transfer's throughput.

An application-based socket buffer auto-tuning technique, called *Dynamic Right-Sizing (DRS)*, has been proposed in [17]. DRS measures the RTT of the path prior to the start of the connection (exogenous RTT). To estimate the bandwidth of the path, DRS measures the average throughput at the receiving side of the application. It is important to note however, that the target transfer throughput does not only depend on the congestion window, but also on the *current* socket buffer size. Thus, DRS will not be able to estimate in general the socket buffer size that maximizes the target transfer's throughput, as it may be limited by the current socket buffer size. The socket buffer sizing objective of DRS does not correspond to one of the BDP definitions in the previous section. A comparison of some socket buffer sizing mechanisms is made in [28].

We finally note that the latest stable version of the Linux kernel (2.4) uses a non-standardized socket buffer sizing algorithm. In particular, even if the application has specified a large receive-socket buffer size (using the *setsockopt* system call), the TCP receiver advertizes a small receive-window that increases gradually with every ACKed segment. It also appears that Linux 2.4 adjusts the send-socket buffer size dynamically, based on the available system memory and the transfer's send-socket buffer backlog.

IV. Under-buffered and over-buffered paths

Let us first examine the effect of network buffering on the throughput of a bulk TCP transfer, in the simple case of *a path that does not carry cross traffic*. Consider the model of §II. What is the average throughput $R(S)$ of a bulk TCP transfer, as a function of the socket buffer size S, the path capacity C, the minimum RTT T_m, and the tight link buffer size B_t? The following result, proven in Appendix-1, shows that we need to consider two cases, depending on the network buffering B_t.

If B_t is less than BDP$_1$, i.e., if $B_t < CT_m$,

$$R(S) = \begin{cases} \frac{S}{T_m} & \text{if } S \leq CT_m \\ C & \text{if } CT_m < S \leq CT_m + B_t \\ \frac{3C}{4}\frac{(B_t+CT_m)^2}{(B_t+CT_m)^2 - B_t CT_m} < C & \text{if } S > CT_m + B_t \end{cases} \tag{4}$$

When S is less than $C \times T_m$, the connection is buffer-limited and it does not saturate the path. As S increases, the throughput $R(S)$ increases linearly until S becomes large enough (CT_m) to saturate the path. If S is sufficiently large to saturate the path, but without causing congestion, i.e., $CT_m < S \leq CT_m + B_t$, the transfer achieves its Maximum Feasible Throughput $\hat{R}$, that is equal to the capacity C. For a larger socket buffer size ($S > CT_m + B_t$), the transfer causes packet losses at the tight link, and it becomes congestion-limited. Its throughput then depends on the amount of network buffering B_t. If $B_t < CT_m$, the transfer cannot saturate the path when $S > CT_m + B_t$ because the 'sawtooth' variations of the congestion window also cause throughput reductions. In other words, the transfer's backlog at the tight link is not enough to absorb the loss-induced reduction of the transfer's send-rate. When $B_t < CT_m$, we say that *the path is under-buffered for the target TCP transfer*. The resulting throughput $R(S)$ depends on the number of dropped packets each time the tight link buffer overflows; the expression in (4) assumes *single packet losses*. In the extreme case that $B_t \approx 0$, the target transfer can only achieve 75% utilization of the tight link.

On the other hand, if $B_t \geq CT_m$ the target transfer can saturate the path even if $S > CT_m + B_t$, assuming again single packet losses:

$$R(S) = \begin{cases} \frac{S}{T_m} & \text{if } S \leq CT_m \\ C & \text{if } S > CT_m \end{cases} \tag{5}$$

The important difference in this case is that losses, and the subsequent send-window reductions, do not decrease the transfer's throughput. This is because the transfer's backlog at the tight link is sufficiently large to keep that link saturated while the send-rate is less than C. When $B_t > CT_m$, we say that *the path is over-buffered for the target transfer*.

Note that whether the path is under-buffered or over-buffered depends on the target transfer's RTT T_0. If the network operator provisions the network buffer size as $B_t = C \times T_0$ based on the RTT T_0, the path will be under-buffered for connections with larger RTT, and over-buffered for connections with smaller RTTs.

The previous result also shows why TCP often performs inefficiently in high bandwidth-delay product paths. Such paths require significant amounts of network buffering at each router/switch interface that may become the tight link of the transfer. The required buffering is at least $C \times T_0$ in terms of bytes, or at least T_0 in terms of seconds. If the interface buffers are dimensioned based on open-loop (non-TCP) traffic, such as in [29], the path can be significantly under-buffered for most TCP traffic. For this reason, some network providers install hundreds of msec worth of buffering in their router interfaces. Large amounts of network buffering, however, can also cause large queueing delays and jitter, affecting the performance of real-time and streaming applications.

Sometimes applications set their socket buffer size to the largest possible value that the operating system allows, in an attempt to maximize the resulting throughput. In that case, however, the transfer may become congestion-limited. Then, if the path is under-buffered for that transfer, the latter will not manage to saturate the path. Even if the path is over-buffered, the target transfer may not be able to get its Maximum Feasible Throughput when it loses several packets at a congestion event.[2]

Equations (4) and (5), however, show an important and positive point that we should focus on: *a TCP transfer can achieve its Maximum Feasible Throughput independent of the network buffering B_t, as long as the socket buffer size is limited in the range $CT_m < S \leq CT_m + B_t$*. That range results in path saturation, but without congestion. To operate at that 'sweet spot', the socket buffer size S must be chosen based on the bandwidth and RTT characteristics of the path, rather than to be blindly set to its maximum possible value.

To illustrate this point, Figure 1 shows the goodput of two successive 1-gigabyte transfer from host bwest.cc.gt.atl.ga.us to host nyu.ron.lcs.mit.edu. The BDP$_1$

[2]Bursty losses are quite common in Drop-Tail buffers [30].

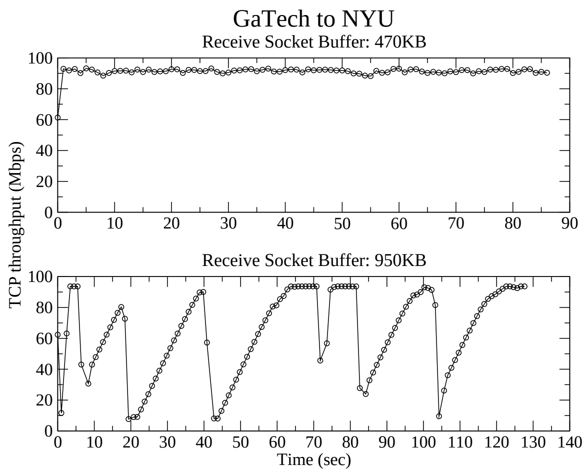

Fig. 1. Throughput of 1-gigabyte transfer with two socket buffer sizes.

of this path is 436KB[3], with a layer-2 capacity of 100Mbps. In the top transfer of Figure 1, the receive-socket buffer size (which is smaller than the send-socket buffer size) is 470KB. With this value of S, the transfer does not experience any packet losses, and its goodput is about 91.3Mbps, close to the capacity of this path. On the other hand, in the bottom transfer of Figure 1, the receive-socket buffer size is set to 950KB, and the transfer receives an average goodput of only 61.7Mbps. The reason is that with the larger socket buffer size, the transfer overloads the network buffers, causing (bursty) packet losses, and subsequent large window reductions. This experiment illustrates that a limited socket buffer size can improve the throughput of a TCP transfer, when it is large enough to saturate the path, but not so large that it would cause congestion.

V. MFT AT A NON-CONGESTED PATH

In this section, we consider *a path that is not congested (i.e., no packet losses at tight link) prior to the target TCP transfer*. Our objective is to examine the relation between the throughput $R(S)$ of the target transfer and its socket buffer size S, and to identify the Maximum Feasible Throughput for different types of cross traffic.

This throughput of the target transfer depends on the *congestion responsiveness* of cross traffic, i.e., the way in which the throughput of cross traffic flows is reduced after the start of the target transfer. We consider three types of cross traffic in a non-congested path:
1. Rate-controlled UDP sources with constant average rate.
2. Buffer-limited persistent TCP transfers ('elephants').
3. Size-limited short TCP transfers ('mice').
As will be shown next, the previous traffic types are fundamentally different in terms of their congestion responsiveness. Even though we do not claim that these are the only types of cross traffic, they represent three distinct and major points in the congestion responsiveness space.

[3]Throughout the paper, KB means 1000 bytes.

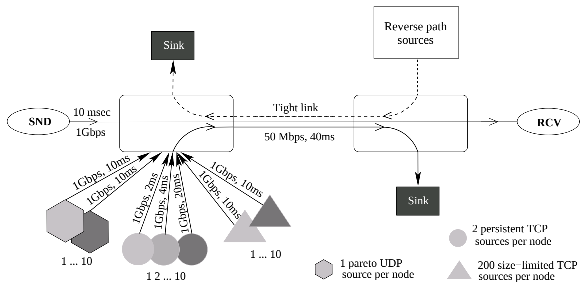

Fig. 2. Simple simulation topology.

The following results are illustrated with NS simulations. The simulated network is shown in Figure 2[4]. All data packets are 1000 bytes long. The structural path characteristics are C=50Mbps, T_m=100msec, and so CT_m=625,000 bytes or 625 data packets. For each cross traffic type, we simulate three tight link buffer sizes B_t: $\frac{1}{2}CT_m$=313 packets (under-buffered), CT_m=625 packets, (well-buffered), and $2CT_m$=1250 packets (over-buffered). Depending on the type of cross traffic that we consider next, some of the sources shown in Figure 2 are turned off. The reverse path carries the same cross traffic as the forward path.

A. *Cross traffic with constant average rate*

Suppose that the tight link of the path carries only rate-controlled cross traffic, with constant average rate $R_c < C_t$. This type of cross traffic will not adapt its send-rate after the target TCP transfer starts, independent of any losses or queueing delays in the path. For this reason, we also refer to it as *congestion unresponsive*. The dynamic characteristics of the path, prior to the target transfer, need to include the additional load, average queueing delay, and buffer occupancy that the cross traffic causes in the tight link of the

[4]We use this simplistic topology here to isolate the cross traffic type from more complex effects that can appear in a multihop topology. A more complex network is simulated in §VII.

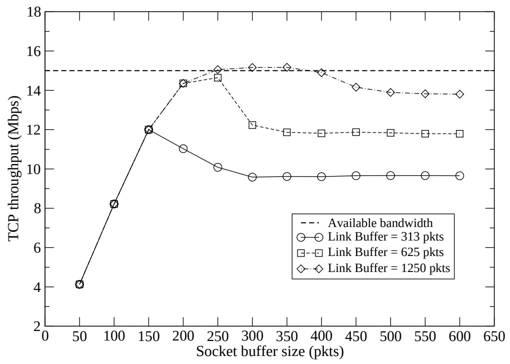

Fig. 3. Cross traffic: rate-controlled UDP (constant average rate).

path. Specifically, the available bandwidth is $A = C - R_c$, the average exogenous RTT is $T_e \geq T_m$, and the average available buffer space at the tight link is $\tilde{B}_t \leq B_t$.

As a first-order approximation, we assume a fluid model for both the cross traffic and the target transfer. Under this assumption, the dynamic characteristics of the path become constant. Following the same derivations as in Appendix-1, the throughput of the target transfer would be then given by Equations (4) and (5), replacing C by A, T_m by T_e, and B_t by $\tilde{B}_t$. So, the path would be over-buffered if $\tilde{B}_t \geq AT_e$, and in that case the target transfer throughput would be $R(S) = S/T_e$ if $S \leq AT_e$, and $R(S) = A$ if $S > AT_e$. If the path is under-buffered (i.e., $\tilde{B}_t < AT_e$), the throughput will drop to $\frac{3A}{4}(\tilde{B}_t + AT_e)^2/[(\tilde{B}_t + AT_e)^2 - \tilde{B}_t AT_e]$ when $S > AT_e + \tilde{B}_t$. So, when the cross traffic has a constant average rate R_c, the MFT of the target transfer is the available bandwidth A, i.e., $\hat{R} = A = C - R_c$. Also, the optimal socket buffer size $\hat{S}$ would be any value in $[AT_e, AT_e + \tilde{B}_t]$.

Because of traffic burstiness however, a backlog will be created at the tight link even when $S < AT_e$, and packet losses can occur even when $S < AT_e + \tilde{B}_t$. As the network buffering B_t decreases, the deviations between the fluid traffic model and bursty traffic would be larger. So, the MFT in practice can be less than A, especially in paths with inadequate network buffering. For the same reason, the optimal socket buffer size in practice would be closer to $\hat{S} = AT_e$ (empty buffer) rather than to $AT_e + \tilde{B}_t$ (full buffer).

Figure 3 shows the target transfer throughput $R(S)$, for three network buffering levels, as a function of the socket buffer size S. The cross traffic is generated from 10 sources with Pareto distributed packet interarrivals (scale parameter α=1.5). The tight link utilization is 70% initially (A=15Mbps), and the exogenous RTT is 102ms. According to the previous model, the optimal socket buffer size would be $\hat{S} = AT_e$=191pkts. When B_t=1250pkts, we see that the target transfer can saturate the path ($\hat{R}$=15Mbps, $\hat{S}$=250-350pkts). For lower network buffering (625pkts and 313pkts), the target transfer has a lower MFT (14.5Mbps and 12Mbps), and a smaller optimal socket buffer size (250pkts and 150pkts) because the tight link buffers overflow before that link is saturated.

In summary, when the cross traffic consists of congestion unresponsive flows with constant average rate, the MFT of the target TCP transfer can be up to the initial available bandwidth A.

B. Buffer-limited TCP transfers

Suppose that the cross traffic is generated from a persistent and buffer-limited TCP transfer with maximum window W_{ct} and RTT T_{ct}. Since the transfer is buffer-limited, it does not create congestion in the path, and its throughput is $R_{ct} = W_{ct}/T_{ct} \leq C$.

In the following derivations, we assume again a fluid traffic model. Before the target TCP transfer starts, the available bandwidth in the path is A=$C - R_{ct} \geq 0$, the exogenous RTT is T_e, and the available tight link buffer space is $\tilde{B}_t \leq B_t$. If the target transfer's socket buffer size is $S \leq AT_e$, it will not saturate the path, and $R(S) = S/T_e \leq A$. If however $AT_e < S \leq R_{max}T_e + \tilde{B}_t$, where R_{max} is the maximum target transfer throughput for which there are no packet losses at the tight link (will be derived later), the target transfer will saturate the tight link and it will create a backlog $Q = S - R(S)T_e$. The backlog Q will increase the RTT of the cross traffic transfer to $T'_{ct} = T_{ct} + Q/C$. The window of that transfer is limited to W_{ct} however, and so its throughput will be now reduced to

$$R'_{ct} = \frac{W_{ct}}{T_{ct} + Q/C} = (C - A)\frac{T_{ct}}{T_{ct} + Q/C} < R_{ct} \quad (6)$$

Thus, the target transfer's throughput will be

$$R(S) = C - R'_{ct} = A\frac{T_{ct} + Q/A}{T_c + Q/C} \geq A \quad (7)$$

which means that the target transfer received some of the bandwidth that was previously utilized by the cross traffic transfer. The MFT of the target transfer is achieved when the latter fills up the available buffer at the tight link, but without causing packet losses,

$$\hat{R} = A\frac{T_{ct} + \tilde{B}_t/A}{T_{ct} + \tilde{B}_t/C} \leq A\frac{T_{ct} + B/A}{T_{ct} + B/C} \quad (8)$$

and so the optimal socket buffer size is $\hat{S} = \hat{R}T_e + \tilde{B}_t$. Also note that the *maximum lossless throughput* of the target transfer R_{max} is equal to the MFT.

Figure 4 shows the target transfer throughput $R(S)$, for three network buffering levels, as a function of the socket buffer size. The buffer-limited persistent transfers are generated from 20 TCP Reno sources. The average RTT of the cross-traffic transfers is 100msec, and their maximum window size is 22 packets so that they create a tight link

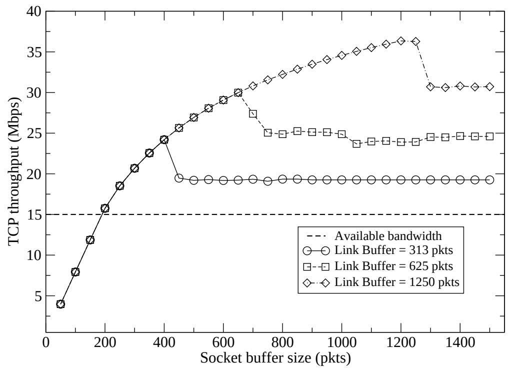

Fig. 4. Cross traffic: buffer-limited persistent TCP transfers.

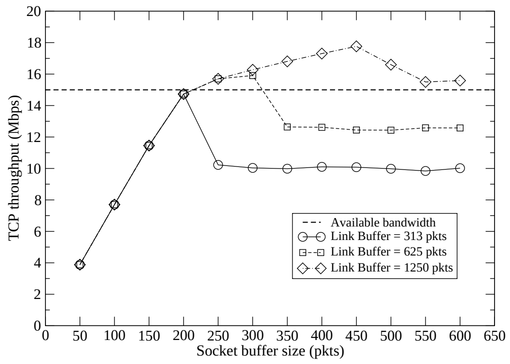

Fig. 5. Cross traffic: size-limited short TCP transfers.

utilization of 70% (A=15Mbps). The measured MFTs are: 24Mbps (B_t=313pkts), 29Mbps (B_t=625pkts), and 36Mbps (B_t=1250pkts). The corresponding MFTs from Equation (8) are 27Mbps, 32Mbps, and 38Mbps. As in §V-A, the model overestimates the MFT and the optimal socket buffer size, because it assumes that losses occur only when the tight link is saturated and the average available buffer space is zero.

In summary, when the cross traffic consists of buffer-limited persistent TCP connections, the MFT of the target TCP transfer is larger than the initial available bandwidth A, and it corresponds to the maximum socket buffer size that does not cause packet losses at the tight link.

C. *Size-limited TCP transfers*

Suppose that the cross traffic is an aggregation of many short TCP transfers ('mice'). Each mouse is a TCP connection, and so it reacts to packet losses by a reduction of its congestion window, and possibly by a timeout. Also, each mouse is window-controlled, which means that RTT increases, due to queueing delays, will also decrease its throughput. These are similar congestion responsiveness characteristics as with buffer-limited TCP transfers ('elephants').

An aggregate of many mice, however, has the additional characteristic that new transfers enter the network constantly over time. The new transfers operate in slow-start, rather than in congestion avoidance, meaning that their window increases exponentially rather than linearly. Also, when congestion occurs, the number of active connections in the network increases because it takes longer for the previously active connections to complete. How would an aggregate of many mice share its utilized bandwidth with the target TCP transfer? This question is hard to model mathematically,[5] and so we limit the analysis of this cross traffic type to simulation results.

Figure 5 shows the target transfer throughput $R(S)$, for three network buffering levels, as a function of the socket buffer size. The mice are generated from 2000 TCP Reno sources (200 at each of 10 nodes) that transfer 10-15 data packets, and then they 'sleep' for a time interval I before starting a new TCP connection. I is uniformly distributed between 4.25 to 4.75 seconds here to achieve a 70% utilization at the tight link (A=15Mbps).

[5] See [31] for some relevant analytical results however.

Note that the throughput function $R(S)$ is similar, in shape, with the corresponding function of buffer-limited cross traffic in Figure 4:
1) $R(S)$ increases linearly with S up to A, until the target transfer saturates the path (if the path is adequately buffered),
2) then, $R(S)$ increases sublinearly with S up to the MFT $\hat{R}$, as the target transfer accumulates backlog at the tight link, increasing the mice's RTT and decreasing their throughput,
3) finally, the target transfer causes packet losses at the tight link, becomes congestion-limited, and its throughput drops to the BTC.

A major difference between mice and elephants, however, is that the MFT with the former cross traffic type is much lower: 14.5Mbps (B_t=313pkts), 16Mbps (B_t=625pkts), and 19.5Mbps (B_t=1250pkts); the corresponding MFT values for elephants were 24Mbps, 29Mbps, and 36Mbps. The MFTs with mice cross traffic are close to the available bandwidth (A=15Mbps), which is generally the case with congestion unresponsive cross traffic. Actually, in the extreme case where the size of each mouse is only a single packet, the aggregate of many mice entering the network with a constant arrival rate would be strictly congestion unresponsive.

In summary, size-limited TCP transfers behave, in terms of their congestion responsiveness, somewhere between buffer-limited persistent TCP transfers and rate-controlled UDP flows: they react individually to losses and increased RTTs, but as an aggregate they do not share much of the bandwidth that they already possess. The MFT with size-limited TCP cross traffic results (as in the case of buffer-limited TCP cross traffic) from the maximum socket buffer size that does not cause packet losses at the tight link.

VI. MFT AT A CONGESTED PATH

In this section, we consider *a path that is congested (i.e., packet losses occur at the tight link) prior to the target TCP transfer*. As in §V, we examine the relation between the throughput $R(S)$ of the target transfer and its socket buffer size S, and identify the Maximum Feasible Throughput for different types of cross traffic.

The key point in the case of a congested path is that the target transfer can experience packet losses caused by cross traffic. This is a consequence of Drop-Tail buffering: dropped packets belong to any flow, rather than only to the flows that cause congestion. So, the target transfer can become congestion-limited not due to its own send-window, but because other flows overload the tight link of the path. A limited socket buffer in this case can only reduce the target

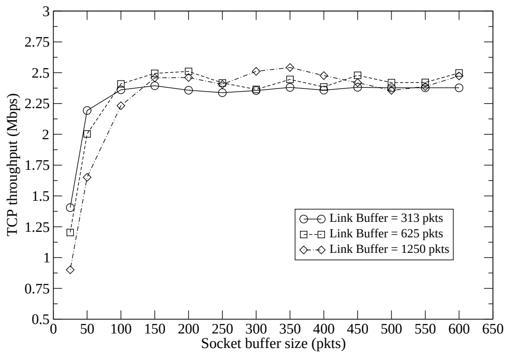

Fig. 6. Cross traffic: congestion-limited persistent TCP transfers.

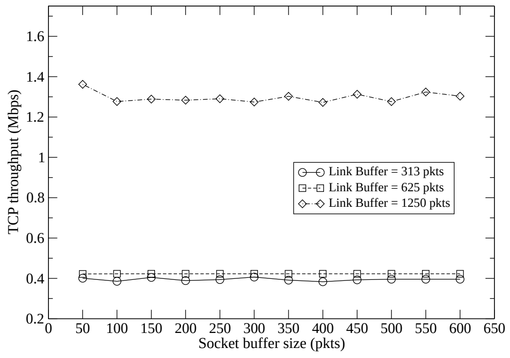

Fig. 7. Cross traffic: size-limited short TCP transfers.

transfer's throughput. Thus, to maximize the target transfer's throughput, the socket buffer size S should be sufficiently large so that the transfer is congestion-limited. Note that this corresponds to BDP$_6$ of §II.

The previous intuitive reasoning can be explained analytically using a result of [32]. Equation (32) of that reference states that the average throughput of a TCP transfer in a congested path with loss rate p and average RTT T is

$$R(S) \approx \min\{\frac{S}{T}, f(T,p)\} \qquad (9)$$

where S is the transfer's maximum possible window (equivalent to socket buffer size), and $f(T,p)$ is a function that depends on TCP's congestion avoidance algorithm. Equation (9) shows that, *in a congested path (p >0), a limited socket buffer size S can only reduce the target transfer's throughput, never increase it.* So, the optimal socket buffer size in a congested path is $\hat{S} = S_\infty$, where S_∞ is a sufficiently large value to make the transfer congestion-limited throughout its lifetime, i.e., $S_\infty > \max W_c$. Also, the MFT in a congested path is the Bulk Transfer Capacity ($\hat{R}$ = BTC) of the target transfer. The BTC can be also predicted from the analytical model of [32] (Equation 20), given the average RTT and loss rate of the target transfer with each tight link buffer size.

The following paragraphs show simulation results for the target transfer throughput $R(S)$, as a function of the socket buffer size, in a congested path for two different cross traffic types. The simulation topology and parameters are as in §V, with the difference that the cross traffic saturates the path ($A \approx 0$), causing packet drops at the tight link, prior to the target transfer. We do not show results for congestion unresponsive traffic, because in that case $R(S) \approx 0$ independent of the socket buffer size.

A. Congestion-limited persistent TCP transfers

Figure 6 shows the target transfer throughput $R(S)$, for three network buffering levels, as a function of the socket buffer size. The congestion-limited persistent transfers are generated from 20 TCP Reno sources. The average RTT of the cross-traffic transfers is 100msec, and their maximum window size is set to a sufficiently large value so that any of them can make the path congested. Prior to the target transfer, the available bandwidth is $A \approx 0$, while the loss rate and average RTT after the target transfer starts is 0.08%-118ms (B_t=313pkts), 0.05%-148ms (B_t=625pkts), and 0.03%-215ms (B_t=1250pkts).

Note the difference between the function $R(S)$ in Figures 4 and 6. In both cases the cross traffic is 20 persistent TCP transfers. In the non-congested path (Figure 4), the cross traffic transfers are buffer-limited, and the target transfer optimizes its throughput with the maximum socket buffer size that does not cause congestion. In that case, limiting the socket buffer size to avoid packet losses increases the target transfer throughput. In a congested path (Figure 6), the cross traffic transfers are congestion-limited. The key observation here is that the target transfer's throughput is optimized when the socket buffer size is sufficiently large to make the transfer congestion-limited. This is the case in these simulation results when the socket buffer size is larger than 100pkts. The MFT is approximately 2.4Mbps, which corresponds to the BTC of the target transfer for the previous values of the RTT and the loss rate.

Note that when the target transfer is buffer limited ($S <$ 100pkts), its throughput is lower for higher values of B_t. This is because a larger B_t allows the cross traffic transfers to introduce more backlog at the tight link, and thus to increase the path's RTT. So, since the target transfer is buffer-limited, its throughput decreases as B_t increases. This is the same effect as in §V-B, but with the role of the buffer-limited cross traffic played by the target transfer.

B. Size-limited short TCP transfers

Figure 7 shows the target transfer throughput $R(S)$, for three network buffering levels, as a function of the socket buffer size. The mice are again generated from 2000 TCP Reno sources that transfer 10-15 data packets, and then 'sleep' for a time interval I before starting a new TCP connection. I varies uniformly between 2 to 2.75 seconds to saturate the tight link of the path ($A \approx 0$). The loss rate and average RTT after the target transfer starts is 1.5%-122ms (B_t=313pkts), 0.85%-148ms (B_t=625pkts), and 0.06%-193ms (B_t=1250pkts).

As in the case of congested-limited persistent TCP transfers, the target transfer optimizes its throughput when it is congested-limited. This is the case in these simulation results for practically any socket buffer size. An interesting difference with Figure 6, however, is that the MFT here is quite lower. The reason is that the aggregation of many short transfers causes a significantly higher loss rate than a few persistent transfers. This is another illustration of the fact that TCP mice are much less congestion responsive than TCP elephants.

VII. SOCKET BUFFER AUTO-SIZING (SOBAS)

In this section we describe SOBAS, an application-layer mechanism that automatically adjusts the socket buffer size of a TCP transfer. SOBAS' objective is to obtain a throughput that is close to the transfer's Maximum Feasible Throughput. There are two key points about SOBAS. First, it does not require changes at the TCP protocol or its implementation, and so, in principle at least, it can be integrated with any TCP-based bulk transfer application. Second, it does not require prior knowledge of the structural or dynamic network path characteristics (such as capacity, available bandwidth, or BTC).

We next state SOBAS' scope and some important underlying assumptions. First, SOBAS is appropriate for persistent (bulk) TCP transfers. Its use would probably not improve the throughput of short transfers that terminate before leaving slow-start. Second, we assume that the TCP implementation at both end-hosts supports window scaling, as specified in [33]. This is the case with most operating systems today [15]. Third, we assume that an application can dynamically change its send and receive socket buffer size during the corresponding TCP transfer, increasing or decreasing it[6]. This is the case in FreeBSD, NetBSD, and Solaris, while Linux 2.4 uses a non-standardized receive-socket buffer tuning algorithm that does not grant the application requests [5]. We are not aware of how other operating systems react to dynamic changes of the socket buffer size. Fourth, we assume that the maximum socket buffer size at both the sender and receiver, normally configured by the system administrator based on the available system memory, is set to a sufficiently large value so that it never limits a connection's throughput. Even though this is not always the case, it is relatively simple to change this parameter in most operating systems [15]. Finally, we assume that the network links use Drop-Tail buffers, rather than RED-like active queues. This is also widely the case.

[6]If an application requests a send-socket buffer decrease, the TCP sender should stop receiving data from the application until its send-window has been decreased to the requested size, rather than dropping data that are already in the send-socket (see [34] §4.2.2.16). Similarly, in the case of a decrease of the receive-socket buffer size, no data should be dropped.

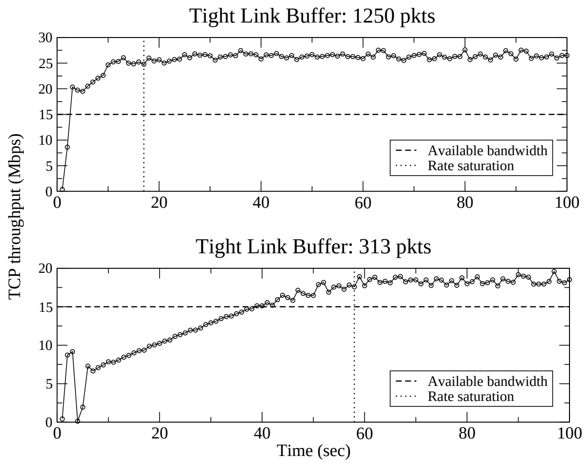

Fig. 8. Receive-throughput using SOBAS for two tight link buffer sizes.

A. Description of SOBAS

The SOBAS receiving-end sends an 'out-of-band' periodic stream of UDP packets to the sending-end. These packets are ACKed by the sender, also with UDP packets. The out-of-band packets, referred to as *periodic probes*, serve two purposes. First, they allow the SOBAS receiver to maintain a running-average of the path's RTT. Second, the receiver can infer whether the forward path is congested, measuring the loss rate of the periodic probes in the forward path. In the current prototype, the size of the periodic probes is 100 bytes, and they are sent with a period of 20ms (overhead rate: 40kbps).

In the case of a non-congested path (see §V), the target transfer reaches its Maximum Feasible Throughput with the largest socket buffer that does not cause packet losses. To detect that point, SOBAS needs to also monitor the goodput at the receiving-end. Specifically, SOBAS measures the transfer's *receive-throughput* R_r at the application-layer, counting the amount of bytes received in every second. SOBAS also knows the initial socket buffer size S at both the sender and the receiver, as well as the running average of the RTT $\bar{T}$, and so it can check whether the receive-throughput R_r is limited by the socket buffer size (i.e., $R_r \approx S/\bar{T}$), or by the congestion window (i.e., $R_r < S/\bar{T}$).

Upon connection establishment, SOBAS sets the send and receive socket buffers to their maximum possible value. So, initially the connection should be congestion-limited, unless if one of the end-hosts does not have enough memory, or if the maximum socket buffer size is too low.

Suppose now that the path is non-congested. The receive-throughput will keep increasing, as the congestion window increases, until it reaches the Maximum Feasible Throughput. We refer to that point as *rate saturation*. At rate sat-

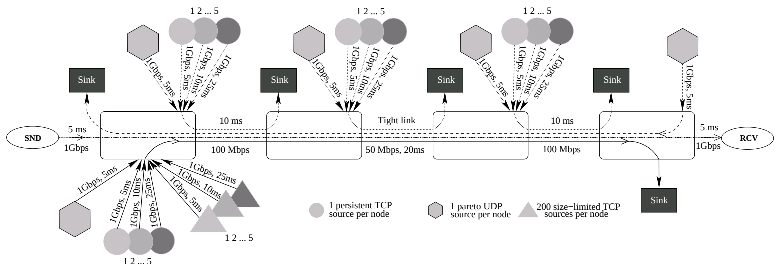

Fig. 9. Multi-hop simulation topology.

uration the receive-throughput 'flattens out', and any further congestion window increases cause queueing at the tight link buffers. The duration of the queue-building period depends on the tight link buffer size, and on whether the TCP sender increases the congestion window multiplicatively (slow-start) or additively (congestion avoidance). If the congestion window is allowed to increase past rate saturation, the tight link buffers will overflow, causing congestion, window reductions, and possibly throughput reductions. SOBAS attempts to avoid exactly that, by limiting the receive-socket buffer size when it detects rate saturation.

The procedure for detecting and reacting to rate saturation is as follows. SOBAS calculates the slope of the last five receive-throughput measurements. A 'rate saturation' event is detected when that slope is approximately zero. Suppose that the receive-throughput at that point is R_{rs} and the corresponding RTT is T_{rs}. SOBAS limits then the receive-socket buffer size to $S_{rs} = R_{rs} \times T_{rs}$. The send-socket buffer size can remain at its previous (maximum possible) value, as it is the smaller of the two socket buffers that limits the transfer's throughput.

In the case of a congested path (see §VI), the target transfer maximizes its throughput when it is congestion-limited, and so S should be large enough to not limit the transfer's send-window. SOBAS checks whether the path is congested only when it has detected rate saturation. At that point, it examines whether any of the last 1000 periodic probes have been lost in the forward path. When that is the case, SOBAS infers that the path is congested, and it does not reduce the socket buffer size.

During slow-start, bursty packet losses can occur because the congestion window increases too fast. Such losses are often followed by one or more timeouts. Also, successive losses can cause a significant reduction of the *ssthresh* parameter, slowing down the subsequent increase of the congestion window. This effect has been studied before (see [8], [10] and references therein), and various TCP modifications have been proposed to avoid it.

SOBAS attempts to avoid the massive losses that can occur in slow-start, imposing an initial limit on the receive-socket buffer size. To do so, SOBAS sends five packet trains at the forward path during the first few round-trips of the transfer. The dispersion of those packet trains is measured at the receiver, and an estimate $\tilde{C}$ of the forward path capacity is quickly made[7]. SOBAS limits the initial socket buffer size to $\tilde{S} = \tilde{C} \times \tilde{T}$, where $\tilde{T}$ is the corresponding RTT estimate at that phase of the transfer. If SOBAS detects that the transfer's window has reached $\tilde{S}$, based on the measured RTT and receive-throughput, the socket buffer size is further increased linearly, by two maximum segments per round-trip, until the detection of rate saturation.

Figure 8 shows simulation results for the receive-throughput of a 1-gigabyte SOBAS transfer at an over-buffered and at an under-buffered path. In the top graph, SOBAS manages to avoid the slow-start losses through the initial socket buffer size limit $\tilde{S}$. Later, SOBAS detects rate saturation when the receive-throughput reaches 24Mbps, and it stops increasing the socket buffer size. In the bottom graph of Figure 8, on the other hand, the tight link is under-buffered and SOBAS fails to avoid the slow-start losses. After the recovery of those losses, TCP increases the congestion window linearly. When the receive-throughput reaches 17.5Mbps, SOBAS detects rate saturation and it sets the socket buffer size to the corresponding send-window. In both cases, SOBAS manages to avoid losses after it has detected rate saturation.

In the current prototype, SOBAS does not attempt to readjust the socket buffer size after it has already done so once at a previous rate saturation event. This approach is justified assuming that the network path's structural and dynamic characteristics are stationary during the TCP transfer. For very long transfers, or when the underlying path tends to change often, it would be possible to modify SOBAS so that it dynamically adjusts the socket buffer size during the transfer; we plan to pursue this approach in future work.

B. Simulation results

We evaluated SOBAS comparing its throughput with the MFT, as well as with the throughput of the six static socket buffer sizing schemes of §II. These comparisons are only meaningful if the underlying network conditions stay the

[7]Even though the dispersion of packet trains cannot be used to accurately estimate the capacity or the available bandwidth of a path [35], it does provides a rough estimate of a path's bandwidth.

same across transfers with different socket buffer sizes. This is only possible with simulations.

Figure 9 shows the multi-hop simulation topology. The target transfer is 1-gigabyte long, and it shares the path with different types of cross traffic. Using the same sources as in §V, the cross traffic mix at the tight link, prior to the target transfer, consists of 60% persistent TCP, 30% mice TCP, and 10% Pareto traffic (in bytes). Table I shows the throughput of the target transfer for the six static socket buffer sizes of §II and for SOBAS, at three different utilizations, and with three tight link buffer sizes B_t. The MFT of the target transfer is also shown.

The first important observation is that *the throughput using SOBAS is close to the Maximum Feasible Throughput, typically within 5-10%*. The deviation of SOBAS from the MFT can be larger however (up to 15-20%) in under-buffered paths (see $B_t = C \times T_m/2$). Under-buffered paths create three problems: first, SOBAS is often unable to avoid the massive losses at the end of slow-start, despite the original limit $\tilde{S}$ of the socket buffer size. Second, under-buffered paths can cause sporadic losses even in moderate loads, especially with bursty traffic. Third, in under-buffered paths, SOBAS is sometimes unable to detect rate saturation before the target transfer experiences packet losses.

A second observation from Table I is that SOBAS provides a higher throughput than the six static socket buffer sizing schemes of §II when the path is non-congested (except in one case). In congested paths, the throughput difference between these schemes, including SOBAS, is minor. The key point, however, is not the actual difference between SOBAS and the sizing schemes of §II. *What is more important is that those schemes require prior knowledge about the capacity, available bandwidth, BTC, or maximum congestion window in the path, while SOBAS determines an appropriate socket buffer size while the transfer is in progress, without any prior information about the path*. Finally, note that the most 'competitive' static socket buffer sizing schemes are the $C \times T_m$ (BDP$_1$), $C \times T_e$ (BDP$_2$), and $S > \max W_c$ (BDP$_6$).

C. Experimental results

We have implemented SOBAS as a simple TCP-based bulk transfer application, and experimented with it at several Internet paths in US and Europe. The dynamic characteristics of an Internet path change over time, and so we are not able to compare SOBAS with other socket buffer sizing schemes, or to measure the MFT, under the same network conditions. Instead, we used our prototype implementation to fine-tune SOBAS, test it in different paths, and see whether it performs robustly 'in the field'.

In Figure 10, we show the goodput of three successive 800MB transfers in a path from host Y to host Z. The capacity of the path is 100Mbps (layer-2), the exogenous RTT is 37ms, and BDP$_1$ is 436KB. The top graph of Figure 10 shows the goodput of the transfer using SOBAS. SOBAS detects rate saturation five seconds after the start of the transfer, and limits the receive-socket buffer size to 559KB. Its average goodput (application layer) is 92.9Mbps.

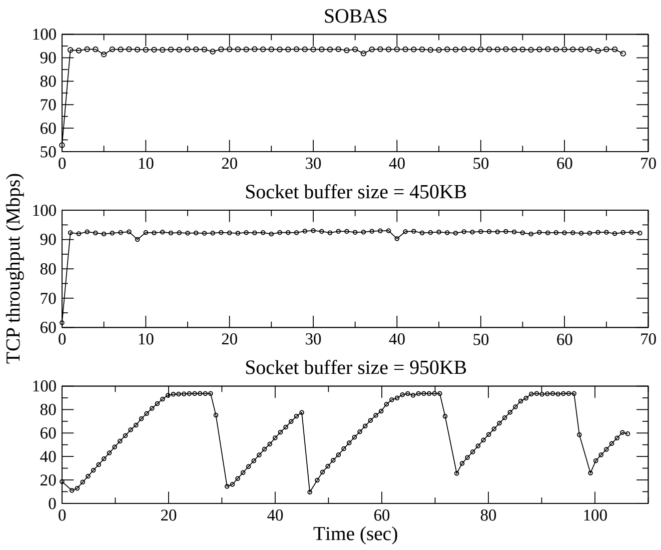

Fig. 10. Throughput of an 800MB transfer from Y to Z with SOBAS, and with two static socket buffer sizes.

The second graph of Figure 10 shows the goodput of the transfer when the socket buffer size is statically set to approximately BDP$_1$ (450KB). With this socket buffer size the transfer also manages to avoid losses, even though its throughput is slightly less than SOBAS (91.3Mbps). An important point is that this socket buffer selection was based on previous knowledge about the capacity and the RTT of the path. SOBAS, on the other hand, did not need this information.

Finally, the third graph of Figure 10 shows the goodput of the transfer when the socket buffer size is statically set to its maximum allowed value at the receiving host (950KB). This choice represents the popular belief in socket buffer sizing that 'larger is better'. Obviously this is not the case! The transfer experiences several bursty losses, resulting in a fairly low average throughput (59.8Mbps).

VIII. Conclusions

This paper considered the problem of TCP socket buffer sizing. We introduced the concept of Maximum Feasible Throughput, as the maximum value of the throughput versus socket buffer size function, and showed that the MFT depends on the amount of network buffering, on the cross traffic type, and on the path's congestion status. We showed that common practices, such as setting the socket buffer size based on a certain definition of the bandwidth-delay product, or simply setting it to a 'big enough' value, often leads to sub-optimal throughput. Finally, we developed

	$C \times T_m$	$C \times T_e$	$A \times T_m$	$A \times T_e$	BTC $\times T_a$	$S > \max W_c$	SOBAS	MFT
Tight link utilization: ρ_t=30% (non-congested path)								
$B_t = C \times T_m/2$	37.6	37.7	29.6	28.9	26.6	29.5	32.9	38.2
$B_t = C \times T_m$	37.6	37.7	29.6	28.9	34.0	31.6	39.1	40.2
$B_t = 2 \times C \times T_m$	37.6	37.7	29.6	28.8	37.7	37.1	39.1	41.7
Tight link utilization: ρ_t=70% (non-congested path)								
$B_t = C \times T_m/2$	15.5	16.3	12.6	12.6	13.5	14.8	17.7	21.5
$B_t = C \times T_m$	24.7	23.7	12.6	12.6	19.0	19.8	25.1	25.6
$B_t = 2 \times C \times T_m$	25.7	25.9	12.6	12.6	24.6	25.9	27.1	29.8
Tight link utilization: ρ_t=100% (congested path)								
$B_t = C \times T_m/2$	2.2	2.2	NA	NA	0.9	2.1	1.9	2.1
$B_t = C \times T_m$	2.3	2.3	NA	NA	1.1	2.3	2.1	2.3
$B_t = 2 \times C \times T_m$	2.8	2.7	NA	NA	1.5	2.7	2.2	2.7

TABLE I

an application-layer mechanism (SOBAS) that can automatically set the socket buffer size close to its optimal value, without prior knowledge of any path characteristics. SOBAS can be integrated with TCP-based bulk transfer applications. It will be more effective, however, if it is integrated with the TCP stack. In that case, the RTT will be known from the corresponding TCP estimates, without requiring UDP-based measurements, and the receive-throughput will be more accurately measurable. Even though SOBAS can avoid causing congestion-related losses, it cannot protect a TCP transfer from random losses. The effect of such losses can be decreased with a limited number of parallel TCP connections. In future work, we plan to integrate SOBAS with the appropriate use of parallel connections.

REFERENCES

[1] PFLDnet, *First International Workshop on Protocols for Fast Long-Distance Networks*, Feb. 2003.

[2] S. Shalunov and B. Teitelbaum, *Bulk TCP Use and Performance on Internet2*, 2002. Also see: http://netflow.internet2.edu/weekly/.

[3] W.-C. Feng, "Is TCP an Adequate Protocol for High-Performance Computing Needs?." Presentation at Supercomputing conference, 2000.

[4] B. Tierney, "TCP Tuning Guide for Distributed Applications on Wide Area Networks," *USENIX & SAGE Login*, Feb. 2001.

[5] T. Dunigan, M. Mathis, and B. Tierney, "A TCP Tuning Daemon," in *Proceedings of SuperComputing: High-Performance Networking and Computing*, Nov. 2002.

[6] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," in *Proceedings of ACM SIGCOMM*, Sept. 1999.

[7] S. Floyd, *HighSpeed TCP for Large Congestion Windows*, Aug. 2002. Internet Draft: draft-floyd-tcp-highspeed-01.txt (work-in-progress).

[8] S. Floyd, *Limited Slow-Start for TCP with Large Congestion Windows*, Aug. 2002. Internet Draft: draft-floyd-tcp-slowstart-01.txt (work-in-progress).

[9] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM*, Aug. 2002.

[10] R. Krishnan, C. Partridge, D. Rockwell, M. Allman, and J. Sterbenz, "A Swifter Start for TCP," Tech. Rep. BBN-TR-8339, BBN, Mar. 2002.

[11] H. Sivakumar, S. Bailey, and R. L. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," in *Proceedings of SuperComputing: High-Performance Networking and Computing*, Nov. 2000.

[12] W. Allcock, J. Bester, J. Bresnahan, A. Chevenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, *gridFTP*, 2000. See http://www.globus.org/datagrid/gridftp.html.

[13] T. J. Hacker and B. D. Athey, "The End-To-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," in *Proceedings of IEEE-CS/ACM International Parallel and Distributed Processing Symposium*, 2002.

[14] L. Cottrell, "Internet End-to-End Performance Monitoring: Bandwidth to the World (IEPM-BW) project," tech. rep., SLAC - IEPM, June 2002. http://www-iepm.slac.stanford.edu/bw/.

[15] M. Mathis and R. Reddy, *Enabling High Performance Data Transfers*, Jan. 2003. Available at: http://www.psc.edu/networking/perf_tune.html.

[16] J. Semke, J. Madhavi, and M. Mathis, "Automatic TCP Buffer Tuning," in *Proceedings of ACM SIGCOMM*, Aug. 1998.

[17] M. K. Gardner, W.-C. Feng, and M. Fisk, "Dynamic Right-Sizing in FTP (drsFTP): Enhancing Grid Performance in User-Space," in *Proceedings IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[18] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in *Proceedings of ACM SIGCOMM*, Aug. 2002.

[19] M. Mathis and M. Allman, *A Framework for Defining Empirical Bulk Transfer Capacity Metrics*, July 2001. RFC 3148.

[20] M. Allman, V. Paxson, and W. Stevens, *TCP Congestion Control*, Apr. 1999. IETF RFC 2581.

[21] L. L. Peterson and B. S. Davie, *Computer Networks, A Systems Approach*. Morgan Kaufmann, 2000.

[22] M. Allman, "Measuring End-to-End Bulk Transfer Capacity," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[23] R. L. Carter and M. E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," *Performance Evaluation*, vol. 27,28, pp. 297–318, 1996.

[24] B. Melander, M. Bjorkman, and P. Gunningberg, "A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks," in *IEEE Global Internet Symposium*, 2000.

[25] J. Guojun, "Network Characterization Service." http://www-didc.lbl.gov/pipechar/, July 2001.

[26] J. Liu and J. Ferguson, "Automatic TCP Socket Buffer Tuning," in *Proceedings of SuperComputing: High-Performance Networking and Computing*, Nov. 2000.

[27] A. Kuzmanovic and E.W.Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer," in *Proceedings of IEEE INFOCOM*, 2003.

[28] E. Weigle and W.-C. Feng, "A Comparison of TCP Automatic Tuning Techniques for Distributed Computing," in *Proceedings IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[29] J. Cao and K. Ramanan, "A Poisson Limit for Buffer Overflow Probabilities," in *Proceedings of IEEE INFOCOM*, June 2002.

[30] V. Paxson, "End-to-end Internet Packet dynamics," in *Proceedings of ACM SIGCOMM*, Sept. 1997.

[31] A. A. Kherani and A. Kumar, "Stochastic Models for Throughput Analysis of Randomly Arriving Elastic Flows in the Internet," in *Proceedings of IEEE INFOCOM*, 2002.

[32] J. Padhye, V.Firoiu, D.Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM*, 1998.

[33] D. Borman, R. Braden, and V. Jacobson, *TCP Extensions for High Performance*, May 1992. IETF RFC 1323.

[34] R. Braden, *Requirements for Internet Hosts – Communication Layers*, Oct. 1989. IETF RFC 1122.

[35] C. Dovrolis, P. Ramanathan, and D. Moore, "What do Packet Dispersion Techniques Measure?," in *Proceedings of IEEE INFOCOM*, pp. 905–914, Apr. 2001.

APPENDIX I: PROOF OF (4) AND (5)

We consider the model of §II, but without any cross traffic in the path. Also, we ignore the initial slow-start phase of the target transfer, and focus on its steady state throughput. We assume that all packet losses are single and due to congestion, and that they are recovered through fast-retransmit. The following derivations are similar to those of [32], with the difference that, when $S > CT_m + B_t$, we derive the RTT variations as a function of the transfer's window.

1. $S \leq CT_m$
Since $S/T_m \leq C$, the transfer cannot saturate the path. Thus, after the initial slow-start, the throughput stays at $R(S) = S/T_m$.

2. $CT_m < S \leq CT_m + B_t$
Since $S/T_m > C$, the transfer overloads the tight link and causes a backlog $Q = S - CT_m$. Because $Q \leq B_t$ there are no losses, and the transfer is buffer-limited. The transfer's RTT is $T_a = T_m + Q/C$, and the throughput is $R(S) = S/T_a = C$.

3. $S > CT_m + B_t$
In this case, the transfer has a sufficiently large window to cause packet losses at the tight link. So, the transfer is congestion-limited, and its send-window is equal to the congestion window W_c. Simplifying TCP's congestion-avoidance algorithm, we assume that W_c is incremented by one segment when the first packet of the previously transmitted window is ACKed. So, we can split the transfer's duration into successive 'rounds', where each round lasts for the round-trip time of the first packet in the previous window. If $W_c(i)$ is the congestion window (in segments) at the i'th round,

$$W_c(i+1) = \begin{cases} W_c(i) + 1 & \text{if no loss at round } i \\ \\ W_c(i)/2 & \text{if (single) loss at round } i \end{cases} \tag{10}$$

Let $T(i)$ be the RTT for the first packet of round i (duration of round i), and $Q(i)$ be the backlog at the tight link at the start of round i. The backlog at the start of round $i+1$ is determined by the window at the previous round: $Q(i+1) = W_c(i) - R(i)T_m$, where $R(i)$ is the transfer's receive-throughput at round i.

Suppose that the first packet of round k is dropped, i.e., $Q(k) = B_t$. Since the tight link was backlogged in the previous round, we have that $R(k-1) = C$ and so

$$W_c(k) = CT_m + B_t + 1 \quad \text{and} \quad W_c(k+1) = W_c(k)/2 \tag{11}$$

If the next packet loss occurs in round $k+m$, we similarly have that $Q(k+m) = B_t$, $W_c(k+m) = CT_m + B_t + 1 = W_c(k+1) + (m-1)$ and so

$$m = \frac{CT_m + B_t - 1}{2} \approx \frac{CT_m + B_t}{2} \tag{12}$$

The approximation is justified when we consider paths with relatively large BDP and/or buffering ($CT_m + B_t \gg 1$).

The number of packets sent in the time period V between the two losses, from round $k+1$ to round $k+m$, is

$$N_V = \sum_{i=1}^{m} W_c(k+i) \approx \frac{3}{8}(CT_m + B_t)^2 \tag{13}$$

Thus, the loss rate p that the transfer causes in its path is

$$p = \frac{1}{N_V} = \frac{8/3}{(CT_m + B_t)^2} \tag{14}$$

To calculate the transfer's throughput, we need to consider the RTT variations with the window size. Specifically, the RTT at round i is $T_i = T_m + Q(i)/C$. The duration of the period V is

$$T_V = \sum_{i=1}^{m} T(k+i) = \sum_{i=1}^{m} [T_m + \frac{Q(k+i)}{C}] \tag{15}$$

3-a. Over-buffered path: $B_t \geq CT_m$
In this case, the minimum window size $W_c(k+1)$ is larger than the path's BDP, because $(CT_m + B_t + 1)/2 > CT_m$. So, the tight link remains backlogged even after the loss, and the backlog will be $Q(k+i) = W_c(k+i-1) - CT_m$ for $i = 1, \ldots m$. Thus, the duration T_V is

$$T_V = mT_m + \sum_{i=1}^{m} [\frac{W_c(k+i-1)}{C} - T_m] = \frac{3}{8C}(CT_m + B_t)^2 \tag{16}$$

We see that $R = \frac{N_V}{T_V} = C$, which verifies that, in the over-buffered case, the transfer saturates the path.

3.b Under-buffered path: $B_t < CT_m$
In this case, the tight link will not be backlogged for n rounds after the loss. $n+1$ rounds after the loss, the backlog will become $Q(k+n+1) = 1$, and $W_c(k+n+1) = 1 + CT_m = W_c(k+1) + n$. Thus, $n = \frac{CT_m - B_t + 1}{2} \approx \frac{CT_m - B_t}{2}$.

The RTT T_i is equal to T_m at rounds $k+1, \ldots k+n$, and equal to $T_m + Q(i)/C$ at rounds $k+n+1, \ldots k+m$. Thus, the duration T_V is

$$T_V = nT_m + (m-n)T_m + \sum_{i=n+1}^{m} [\frac{W_c(k+i-1)}{C} - T_m] =$$

$$\frac{(CT_m)^2 + CT_m(B_t+2) + B_t(B_t+2)}{2C} \approx$$

$$\frac{(CT_m)^2 + CT_mB_t + B_t^2}{2C}$$

So, the transfer's average throughput will be

$$R = \frac{N_V}{T_V} = \frac{3C}{4}\frac{(CT_m+B_t)^2}{(CT_m+B_t)^2 - CT_mB_t} < C. \qquad (17)$$