Socket Buffer Auto-Sizing for High-Performance Data Transfers

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

Abstract—*It is often claimed that TCP is not a suitable transport protocol for data intensive Grid applications in high-performance networks. We argue that this is not necessarily the case. Without changing the TCP protocol, congestion control, or implementation, we show that an appropriately tuned TCP bulk transfer can saturate the available bandwidth of a network path. The proposed technique, called SOBAS, is based on automatic socket buffer sizing at the application layer. In non-congested paths, SOBAS limits the socket buffer size based on direct measurements of the received throughput and of the corresponding round-trip time. The key idea is that the send window should be limited, after the transfer has saturated the available bandwidth in the path, so that the transfer does not cause buffer overflows ("self-induced losses"). A difference with other socket buffer sizing schemes is that SOBAS does not require prior knowledge of the path characteristics, and it can be performed while the transfer is in progress. Experimental results in several high bandwidth-delay product paths show that SOBAS provides consistently a significant throughput increase (20% to 80%) compared to TCP transfers that use the maximum possible socket buffer size. We expect that SOBAS will be mostly useful for applications such as GridFTP in non-congested wide-area networks.*

Keywords: Grid computing and networking, TCP throughput, available bandwidth, bottleneck bandwidth, fast long-distance networks.

I. Introduction

The emergence of the Grid computing paradigm raises new interest in the end-to-end performance of data intensive applications. In particular, the scientific community pushes the edge of network performance with applications such as distributed simulation, remote colaboratories, and frequent multigigabyte transfers. 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 for most of the time. Additionally, through the deployment of Gigabit and 10-Gigabit Ethernet interfaces, congestion also becomes rare at network edges and end-hosts. With all this bandwidth, it is not surprising that Grid users expect superb end-to-end performance. However, this is not always the case. A recent measurement study at Internet2 showed that 90% of the bulk TCP transfers (i.e., more than 10MB) receive less than 5Mbps [1].

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), and by an equipment donation from Intel Corporation.

It is widely believed that a major reason for the relatively low end-to-end throughput is TCP. This is either due to TCP itself (e.g., congestion control algorithms and parameters), or because of local system configuration (e.g., default or maximum socket buffer size) [2]. TCP is blamed that it is slow in capturing the available bandwidth of high performance networks, mostly because of two reasons:
1. Small socket buffers at the end-hosts limit the effective window of the transfer, and thus the maximum throughput.
2. Packet losses cause large window reductions, with a subsequent slow (linear) window increase rate, reducing the transfer's average throughput.
Other TCP-related issues that impede performance are multiple packet losses at the end of slow start (commonly resulting in timeouts), the inability to distinguish between congestive and random packet losses, the use of small segments, or the initial *ssthresh* value [3], [4].

Researchers have focused on these problems, pursuing mostly three approaches: TCP modifications [5], [6], [7], [8], [9], [10], parallel TCP transfers [11], [12], and automatic buffer sizing [3], [13], [14], [15]. Changes in TCP or new congestion control schemes, possibly with cooperation from routers [7], can lead to significant benefits for both applications and networks. However, modifying TCP has proven to be quite difficult in the last few years. Parallel TCP connections can increase the aggregate throughput that an application receives. This technique raises 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. Finally, techniques that automatically adjust the socket buffer size can be performed at the application-layer, and so they do not require changes at the TCP implementation or protocol. In this work, we adopt the automatic socket buffer sizing approach.

How is the socket buffer size related to the throughput of a TCP connection? The send and receive socket buffers

should be sufficiently large so that the transfer can saturate the underlying network path. Specifically, suppose that the bottleneck link of a path has a transmission capacity of C bps and the path between the sender and the receiver has a Round-Trip Time (RTT) of T sec. When there is no competing traffic, the connection will be able to saturate the path if its send window is $C \times T$, i.e., the well known *Bandwidth Delay Product (BDP)* of the path. For the window to be this large, however, TCP's flow control requires that the smaller of the two socket buffers (send and receive) should be equally large. If the size S of the smaller socket buffer is less than $C \times T$, the connection will underutilize the path. If S is larger than $C \times T$, the connection will overload the path. In that case, depending on the amount of buffering in the bottleneck link, the transfer may cause buffer overflows, window reductions, and throughput drops.

The BDP and its relation to TCP throughput and socket buffer sizing are well known in the networking literature [16]. As we explain in §II, however, the socket buffer size should be equal to the BDP *only when the network path does not carry cross traffic*. The presence of cross traffic means that the "bandwidth" of a path will not be C, but somewhat less than that. Section II presents a model of a network path that helps to understand these issues, and it introduces an important measure referred to as Maximum Feasible Throughput (MFT).

Throughout the paper, we distinguish between congested and non-congested network paths. In the latter, the probability of a congestive loss (buffer overflow) is practically zero. Non-congested paths are common today, especially in high-performance well provisioned networks. In §III, we explain that, in a non-congested path, a TCP transfer can saturate the available bandwidth as long as it does not cause buffer overflows. To avoid such *self-induced losses*, we propose to limit the send window using appropriately sized socket buffers. In a congested path, on the other hand, losses occur independent of the transfer's window, and so limiting the latter can only reduce the resulting throughput.

The main contribution of this paper is to *develop an application-layer mechanism that automatically determines the socket buffer size that saturates the available bandwidth in a network path, while the transfer is in progress*. Section IV describes this mechanism, referred to as *SOBAS (SOcket Buffer Auto-Sizing)*, in detail. SOBAS is based on direct measurements of the received throughput and of the corresponding RTT at the application layer. The key idea is that the send window should be limited, after the transfer has saturated the available bandwidth in the path, so that the transfer does not cause buffer overflows, i.e., to avoid self-induced losses. In congested paths, on the other hand, SOBAS disables itself so that it does not limit the transfer's window. We emphasize that *SOBAS does not require changes in TCP*, and that it can be integrated with any TCP-based bulk data transfer application, such as GridFTP [17].

Experimental results in several high BDP paths, shown in §V, show that SOBAS provides consistently a significant throughput increase (20% to 80%) compared to TCP transfers that use the maximum possible socket buffer size. A key point about SOBAS is that it does not require prior knowledge of the path characteristics, and so it is simpler to use than socket buffer sizing schemes that rely on previous measurements of the capacity or available bandwidth in the path. We expect that SOBAS will be mostly useful for applications such as GridFTP in non-congested wide-area networks.

In §VI, we review various proposals for TCP optimizations targeting high BDP paths, as well as the previous work in the area of socket buffer sizing. We finally conclude in §VII.

II. Socket Buffer Size and TCP Throughput

Consider a unidirectional TCP transfer from a sender $\mathcal{SND}$ to a receiver $\mathcal{RCV}$. TCP uses window based flow control, 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* [18], 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., $W_r \leq B_r$. In the rest of this paper, we assume that W_r=B_r, i.e., the receiving application is sufficiently fast to consume any delivered data, keeping the receive socket buffer always empty. The send window is then limited by:

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

where $S = \min\{B_s, B_r\}$ *is the smaller of the two socket buffer sizes.*

If the send window W_s is limited by W_c we say that the transfer is *congestion limited*, while if it is limited by S, we say that the transfer is *buffer limited*. 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 *target transfer*; the rest of the traffic in $\mathcal{P}$ 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 path transmits packets with a *capacity* of C_i bps. Arriving packets are discarded in a Drop Tail manner. 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 $A_i = C_i \times (1 - \rho_i)$. Adopting the terminology of [19], we refer to the link of the forward path $\mathcal{P}_f$ with the minimum available bandwidth $A = \min_{\mathcal{P}_f}\{A_i\}$ as the *tight link*. The buffer size of the tight link is denoted by B_t.

A link is *saturated* when its available bandwidth is zero. Also, a link is *non-congested* when its packet loss rate due to congestion is practically zero; otherwise the link is *congested*. For simplicity, we assume that the only congested link in the forward path is the tight link. A path is called *congested* when its tight link is congested; otherwise, the path is called *non-congested*.

The *exogenous RTT* T_e of the path is the sum of all average delays along the path, including both propagation and queueing delays, *before the target transfer starts*. The *average RTT* T_a, on the other hand, is the sum of all average delays along the path *while the target transfer is in progress*. In general, $T_a \geq T_e$ due to increased queueing caused by the target transfer.

From Equation (3), we can view the target transfer throughput as a function $R(S)$. Then, an important question is: *given a network path $\mathcal{P}$, what is the value(s) $\hat{S}$ of the socket buffer size that maximizes the target transfer throughput $R(S)$?* We refer to the maximum value of $R(S)$ as the *Maximum Feasible Throughput (MFT)* $\hat{R}$. The conventional wisdom, as expressed in textbooks [16], operational handouts [2], and research papers [14], is that the socket buffer size $\hat{S}$ should be equal to the Bandwidth Delay Product of the path, where "bandwidth" is the capacity of the path C, and "delay" is the exogenous RTT of the path T_e, i.e., $\hat{S} = C \times T_e$. Indeed, if the send window is $W_s = C \times T_e$, and *assuming that there is no cross traffic in the path*, the tight link becomes saturated (i.e., A=0) but not congested, and so the target transfer achieves its MFT ($\hat{R} = C$).

In practice, a network path always carries some cross traffic, and thus $A < C$. If $S > A \times T_e$, the target transfer will saturate the tight link, and depending on B_t, it may also cause packet losses. Losses, however, cause multiplicative drops in the target transfer's send window, and, potentially, throughput reductions. 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 an additional important implication. If the cross traffic is TCP (or TCP friendly), it will react to the presence of the target transfer reducing its rate, either because of packet losses, or because the target transfer has increased the RTT in the path ($T_a > T_e$). In that case, the target transfer can achieve a higher throughput than the initial available bandwidth A. In other words, the MFT can be larger than the available bandwidth, depending on the *congestion responsiveness* of the cross traffic.

The previous discussion reveals several important questions. What is the optimal socket buffer size $\hat{S}$ and the MFT in the general case of a path that carries cross traffic? What is the relation between the MFT and the available bandwidth A? How is the MFT different in congested versus non-congested paths? How should a socket buffer sizing scheme determine $\hat{S}$, given that it does not know a priori A and B_t? These questions are the subject of the next section.

III. Maximum Feasible Throughput and Available Bandwidth

A. Non-congested paths

Suppose first that the network path $\mathcal{P}$ is non-congested. We illustrate next the effect of the socket buffer size S on the throughput $R(S)$ with an example of actual TCP transfers in an Internet2 path.

The network path is from a host at Ga-Tech (regulus.cc.gatech.edu) to a RON host at NYU (nyu.ron.lcs.mit.edu) [20]. The capacity of the path is C=97Mbps[1], the exogenous RTT is T_e=40ms, and the loss rate that we measured with *ping* was zero throughout our experiments. We repeated a 200MB TCP transfer four times with different values of S. Available bandwidth measurements with *pathload* [19] showed that A was practically constant before and after our transfers, with $A \approx$ 80Mbps.

Figure 1 shows the throughput and RTT of the TCP connection when S=128KB. In this case, the throughput of the transfer remains relatively constant, the connection does not experience packet losses, and the transfer is buffer limited. The transfer does not manage to saturate the path because $R(S)=S/T_e$=25.5Mbps, which is much less than A. Obviously, any socket buffer sizing scheme that sets S to less than A/T_e will lead to poor performance.

Next, we increase S to the value that is determined by the available bandwidth, i.e., $S=A \times T_e$=400KB (see Figure 2). We expect that in this case the transfer will saturate the path, without causing persistent queueing and packet losses. Indeed, the connection is still buffer limited, getting approximately the available bandwidth in the path ($R(S) \approx$79Mbps). Because S was determined by the available bandwidth, the transfer did not introduce a persistent backlog in the queue of the tight link, and so $T_a = T_e$=40ms.

One may think that the previous case corresponds to the optimal socket buffer sizing, i.e., that the MFT is $\hat{R} = A$.

[1] The capacity and available bandwidth measurements mentioned in this paper refer to the IP layer. All throughput measurements, on the other hand, refer to the TCP layer.

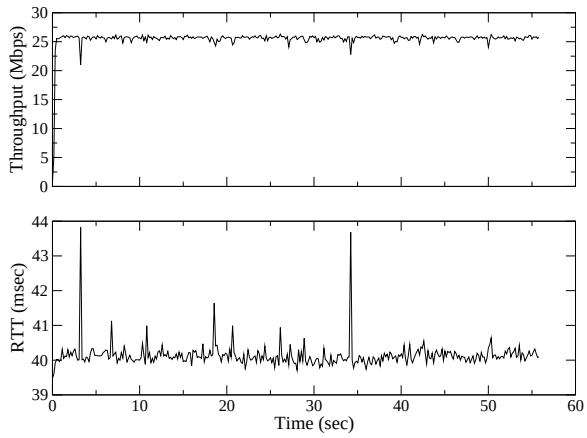

Fig. 1. Throughput and RTT of a 200MB transfer with S=128KB.

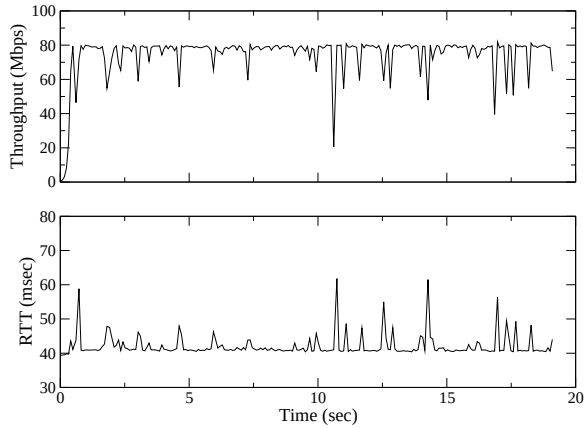

Fig. 2. Throughput and RTT of a 200MB transfer with S=400KB.

The MFT of a path, however, depends on the *congestion responsiveness of the cross traffic*. If the cross traffic is not congestion responsive, such as unresponsive UDP traffic or an aggregate of short TCP flows, it will maintain an almost constant throughput as long as the target transfer does not cause buffer overflows and packet losses. In this case, the MFT will be equal to the available bandwidth. If the cross traffic consists of buffer limited persistent TCP transfers, however, any increase in the RTT will lead to reduction of their throughput. In that case, the target transfer can "steal" some of the throughput of cross traffic transfers by causing a persistent backlog in the tight link, making the MFT larger than A. An analysis of the congestion responsiveness of Internet traffic is outside the scope of this paper; interested readers can find some results in [21].

To illustrate the effect of congestion responsiveness of cross-traffic on MFT, we further increase S to 550KB (see Figure 3). The first point to note is that the transfer is still buffer limited, as it does not experience any packet losses. Second, the RTT increases by 9ms from T_e=40ms to

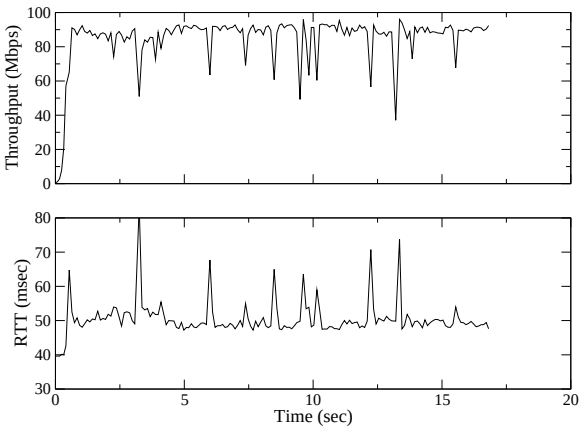

Fig. 3. Throughput and RTT of a 200MB transfer with S=550KB.

T_a=49ms. Consequently, the throughput of the target transfer reaches $R(S)$=S/T_a≈90Mbps, which is more than the available bandwidth before the target transfer. Where does this additional throughput come from? Even though we do not know the nature of cross traffic in this path, we can assume that some of the cross traffic flows are buffer limited TCP flows. The throughput of such flows is inversely proportional to their RTTs, and so the 9ms RTT increase caused by the target transfer leads to a reduction of their throughput.

One may think that increasing S even more will lead to higher throughput. That is not the case however. If we increase S beyond a certain point, the target transfer will cause buffer overflows in the tight link. The transfer will then become congestion limited, reacting to packet drops with large window reductions and slow window increases. To illustrate this case, Figure 4 shows what happens to the target transfer when S is set to 900KB (the largest possible socket buffer size at these end-hosts). The connection experiences several losses during the initial slow-start (about one second after its start), which are followed by a subsequent timeout. Additional losses occur after about 12 seconds, also causing a significant throughput reduction.

The previous four cases illustrate that socket buffer sizing has a major impact on TCP throughput in non-congested paths. The target transfer can reach its MFT with the maximum possible socket buffer size that does not cause self-induced packet losses. We also show that, depending on the congestion responsiveness of cross traffic, *the MFT may be only achievable if the target transfer introduces a persistent backlog in the tight link and a significant RTT increase*. Limiting the socket buffer size based on the available bandwidth, on the other hand, does not increase the RTT of the path but it may lead to suboptimal throughput.

How can a socket buffer sizing scheme determine the optimal value of S for a given network path? An important point is that end-hosts do not know the amount of buffering at the

tight link B_t or the nature of the cross traffic. Consequently, *it may not be possible to predict the value of S that will lead to self-induced losses, and consequently, to predict the MFT.*

Instead, it is feasible to determine S based on the available bandwidth. That is simply *the point in which the received throughput becomes practically constant, and the RTT starts to increase*. Even though setting S based on the available bandwidth may be suboptimal compared to the MFT, we think that it is a better objective for the following reasons:

1. Since the amount of buffering at the tight link is unknown, accumulating a persistent backlog can lead to early congestive losses, reducing significantly the target transfer's throughput.
2. A significant RTT increase can be detrimental for the performance of real-time and interactive traffic in the same path.
3. Increasing the target transfer's throughput by deliberately increasing the RTT of other TCP connections can be considered as an unfair congestion behavior.

B. Congested paths

A path can be congested, for instance, if it carries one or more congestion limited persistent TCP transfers, or if there are packet losses at the tight link due to bursty cross traffic.

The key point that differentiates congested from non-congested paths is that the target transfer can experience packet losses independent of its socket buffer size. This is a consequence of Drop Tail queueing: *dropped packets can belong to any flow*. A limited socket buffer, in this case, can only reduce the target transfer's throughput. So, *to maximize the target transfer's throughput, the socket buffer size S should be sufficiently large so that the transfer is always congestion limited*.

The previous intuitive reasoning can also be shown analytically using a result of [22]. 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

$$R(S) \approx \min\{\frac{S}{T}, f(T,p)\} \quad (4)$$

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 (4) 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., $S_\infty > \max W_c$.

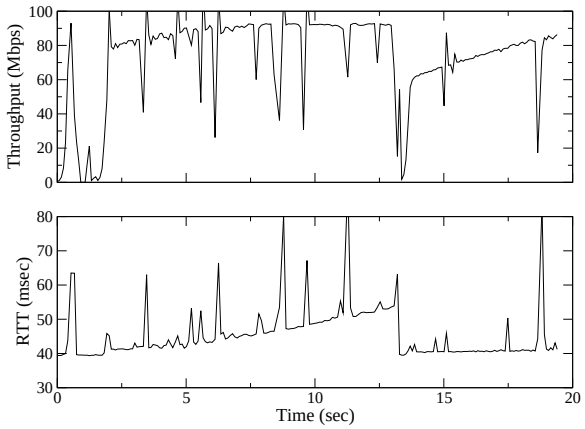

Fig. 4. Throughput and RTT of a 200MB transfer with S=900KB (max).

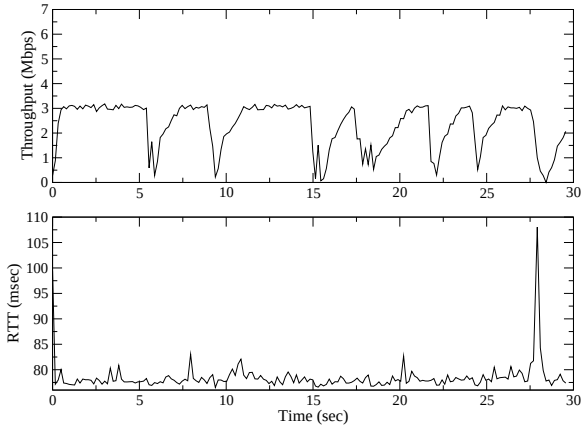

Fig. 5. Throughput and RTT of a 30MB transfer in a congested path with S=30KB.

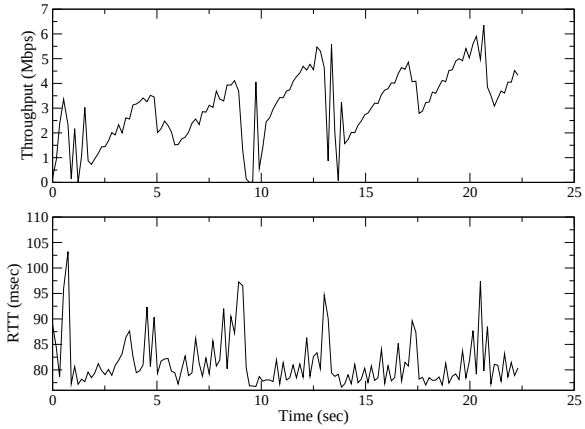

Fig. 6. Throughput and RTT of a 30MB transfer in a congested path with S=S_∞.

To illustrate what happens in congested paths, Figures 5 and 6 show the throughput and RTT of a TCP transfer in a path from *regulus.cc.gatech.edu* to *aros.ron.lcs.mit.edu* at MIT. The capacity of the path is C=9.7Mbps, the RTT is T_e=78ms, while the available bandwidth A is about 3Mbps.

In Figure 5, S is limited to 30KB, which is the value determined by the available bandwidth (S=AT_e). Even though the transfer does not overload the path (notice that the RTT does not show signs of persistent increase) the connection experiences several packet losses. The average throughput of the transfer in this case is 2.4Mbps.

In Figure 6, on the other hand, S is increased to the maximum possible value, and so the transfer is always congestion limited. The transfer experiences again multiple loss events, but since this time it is not limited by S it achieves a larger average throughput, close to 3.1Mbps.

IV. SOcket Buffer Auto-Sizing (SOBAS)

In this section we describe SOBAS. As explained in the previous section, the objective of SOBAS is to saturate the available bandwidth of a non-congested network path, without causing a significant RTT increase. SOBAS does not require changes at the TCP protocol or implementation, and so it can be integrated with any bulk transfer application. It does not require prior knowledge of the capacity or available bandwidth, while the RTT and the presence of congestive losses are inferred directly by the application using UDP out-of-band probing packets. The throughput of the transfer is also measured by the application based on periodic measurements of the received goodput.

We anticipate that SOBAS will be mostly useful in specific application and network environments. First, SOBAS is designed for bulk data transfers. It would probably not improve the throughput of short transfers, especially if they terminate before the end of slow start. Second, SOBAS takes action only in non-congested paths. In paths with persistent congestion or limited available bandwidth, SOBAS will disable itself automatically by setting the socket buffer size to the maximum possible value[2]. Third, SOBAS adjusts the socket buffer size only once during the transfer. This is sufficient as long as the *cross traffic is stationary*, which may be not be a valid assumption if the transfer lasts for several minutes [23]. For extremely large transfers, the application can split the transferred file in several segments and transfer each of them sequentially using different SOBAS sessions.

We next state certain host and router requirements for SOBAS to work effectively. First, the TCP implementation at both end hosts must support window scaling, as specified in [24]. Second, the operating system should allow dynamic changes in the socket buffer size during the TCP transfer, increasing or decreasing it.[3] Third, the maximum allowed socket buffer size at both the sender and the receiver must be sufficiently large so that it does not limit the connection's throughput. Finally, the network elements along the path are assumed to use Drop Tail buffers, rather than active queues. All previous requirements are valid for most operating systems [13] and routers in the Internet today.

[2]The maximum possible socket buffer size at a host can be modified by the administrator.

[3]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 [25] §4.2.2.16). Similarly, in the case of a decrease of the receive socket buffer size, no data should be dropped.

A. Basic idea

The basic idea in SOBAS is the following. In non-congested paths, SOBAS should limit the receiver socket buffer size, and thus the maximum possible send window, so that the transfer saturates the path but does not cause buffer overflows. In congested paths, on the other hand, SOBAS should set the socket buffer size to the maximum possible value, so that the transfer is congestion limited.

SOBAS detects the point in which the transfer has saturated the available bandwidth using two "signatures" in the receive throughput measurements: **flat-rate** and **const-rate-drop**. The **flat-rate** condition is detected when the receive throughput appears to be almost constant for a certain time period. The **const-rate-drop** condition occurs when SOBAS is unable to avoid self-induced losses, and it is detected as a rate drop following a short time period in which the throughput was constant. The detection of these two signatures is described later in more detail.

*if (***flat-rate** *or* **const-rate-drop***)*
{
if (non-congested path)
$S = R \times T$*; (set socket buffer size to rcv-throughput times RTT)*
else
$S = S_{MAX}$*; (set socket buffer size to maximum value)*
}

Fig. 7. Basic SOBAS algorithm.

B. Implementation details and state diagram

Several important details about the SOBAS algorithm are described next.

How does the receiver infer whether the path is congested, and how does it estimate the RTT? The receiver sends an out-of-band periodic stream of UDP packets to the sender. The sender echoes the packets back to the receiver with a probe sequence number. The receiver uses these sequence numbers to estimate the loss rate at the forward path, inferring whether the path is congested or not. Even though it is well-known that periodic probing may not result in accurate loss rate estimation, notice that SOBAS only needs to know whether the path is congested, i.e., whether the loss rate is non-zero. The receiver remembers lost probes in the last 100 UDP probes. If more than one probe was lost, it infers the path to be **congested**; otherwise the path is **non-congested**. Additionally, the periodic path probing allows the receiver to maintain a running average of the RTT. Probing packets

are 100 bytes and they are sent every 10ms resulting in a rate of 80kbps. This overhead is insignificant compared to the throughput benefits that SOBAS provides, as shown in §V. However, this probing overhead means that SOBAS would not be useful in very low bandwidth paths, such as those limited by dial-up links.

How often should SOBAS measure the receive throughput R? SOBAS measures the receiver throughput periodically at the application layer, as the ratio of the amount of bytes received in successive time windows of length Δ=2×RTT. The measurement period Δ is important. If it is too small, and especially if it is smaller than the transfer's RTT, the resulting throughput measurements will be very noisy due to delays between the TCP stack and the application layer, and also due to the burstiness of TCP self-clocking. If Δ is too large, on the other hand, SOBAS will not have enough time to detect that it has saturated the available bandwidth before a buffer overflow occurs. In the Appendix, we derive expressions for the *Buffer Overflow Latency*, i.e., for the amount of time it takes for a network buffer to fill up in two cases: when the target transfer is in slow-start and when it is in congestion-avoidance. Based on those results, we argue that the choice Δ=2×RTT is a reasonable trade-off in terms of accuracy and measurement latency.

How does the receiver detect the two conditions **flat-rate** and **const-rate-drop**? The **flat-rate** condition is true when five successive throughput measurements are almost equal, i.e., when the throughput has become constant.[4] In the current implementation, throughput measurements are considered almost equal if their slope with respect to time is less than half the corresponding throughput increase due to congestion window (cwnd) increases. In congestion-avoidance with Delayed-ACKs, the throughput increase due to cwnd increases is about half MSS per RTT per RTT. At the **flat-rate** point, any further increases in the send window cause persistent backlog in the tight link and RTT increases. The **const-rate-drop** condition is true, on the other hand, when the receive throughput has dropped significantly (by more than 20%) after a period of time (two throughput measurements) in which it was almost constant (within 5%). The condition **const-rate-drop** takes place when SOBAS does not manage to limit the send window before the target transfer experiences a packet loss. This loss is sometimes unavoidable in practice, especially in underbuffered paths. However, SOBAS will avoid any further such losses by limiting the receiver socket buffer after the first loss.

Before the connection is established, SOBAS sets the send and receive socket buffers to their maximum values in order to have a sufficiently large window scale factor. The value of *ssthresh* becomes then equally large, and so the initial slow-start can lead to multiple packet losses. Such losses often result in one or more timeouts, and they can also cause a significant reduction of *ssthresh*, slowing down the subsequent increase of the congestion window. This effect has been studied in [9] and [26]. SOBAS attempts to avoid massive slow start losses using a technique that is similar with that of [9]. The basic idea is to initially limit the receive socket buffer size based on a rough capacity estimate C' of the forward path. C' results from the average dispersion of five packet trains, using UDP probing packets [27]. If the transfer becomes *buffer limited* later on, SOBAS increases periodically the socket buffer size by one Maximum Segment Size (MSS) in every RTT. This linear increase is repeated until one of the **flat-rate** or **const-rate-drop** conditions becomes true.

Figure 8 shows the state diagram of the complete SOBAS algorithm. A couple of clarifications follow. First, State-2 represent the initial slow-start phase. SOBAS can also move out of slow-start if it observes a rate drop without the **flat-rate** signature or without being *buffer limited*. This is shown by the transition from State-2 to State-3 in Figure 8. That transition can take place due to losses before the path has been saturated. Second, State-6 represents the final state in a congested path, which is inferred when SOBAS observes losses in the periodic UDP probes. and it can be reached from states 2, 3 or 4. Overall, the implementation of the algorithm is roughly 1,000 lines of C code.

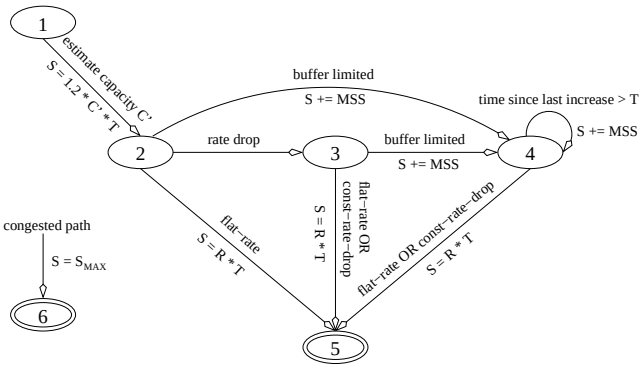

Fig. 8. SOBAS state diagram.

V. Experimental results

We have implemented SOBAS as a simple TCP-based data transfer application. The prototype has been tested over a large number of paths and at several operating systems (including Linux 2.4, Solaris 8, and FreeBSD 4.7). In this section, we present results from a few Internet paths, covering an available bandwidth range of 10-1000Mbps. These paths traverse links in the following networks: Abilene, SOX, ESNet, NYSERNet, GEANT, SUNET (Sweden), and campus networks at the location of the end-hosts (Georgia Tech, LBNL, MIT, NYU, Lulea University). For each path, we compare the throughput that results from SOBAS with the

[4] The required constant throughput measurements are only two, instead of five, when the transfer is in the initial slow-start phase (states 1 and 2 in Figure 8).

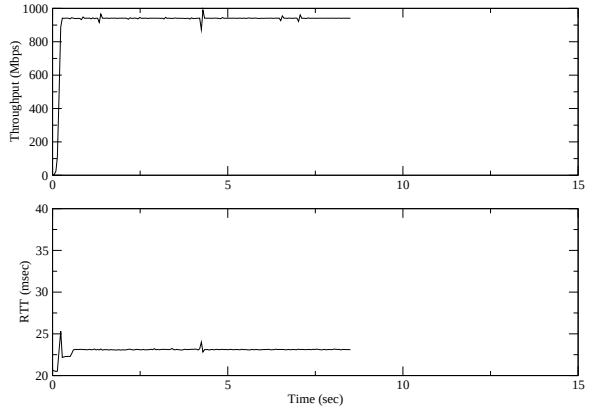

Fig. 9. With SOBAS: Gigabit path, no cross traffic (A=950Mbps).

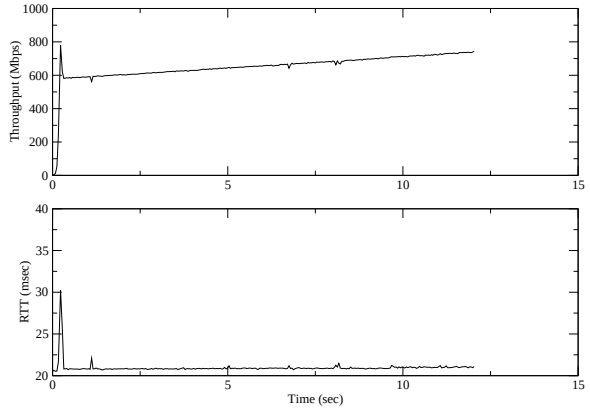

Fig. 10. Without SOBAS: Gigabit path, no cross traffic (A=950Mbps).

throughput that results from using the maximum allowed socket buffer size (referred to as "non-SOBAS"). The latter is what data transfer applications do in order to maximize their throughput. The SOBAS and non-SOBAS transfers on each path are performed in close sequence.

We classify the following paths in three groups, depending on the underlying available bandwidth. The "gigabit path" is located in our testbed and is limited by a Gigabit-Ethernet link (1000Mbps). The "high-bandwidth paths" provide 400-600Mbps and they are probably limited by OC12 links or rate-limiters. The "typical paths" are limited by Fast Ethernet links, and they provide less than 100Mbps. The transfer size is 1GB in the gigabit and the high-bandwidth paths, and 200MB in the typical paths.

A. Gigabit path

Our gigabit testbed consists of four hosts with GigE NICs connected to two Gigabit switches (Cisco 3550). The GigE link between the two switches is the tight link with capacity C=970Mbps at the IP layer. Two hosts (single processor, 2.4GHz Intel Xeon, 1GB RAM, PCI bus 66MHz/64bit, Redhat 7.3) are used as the source and sink of cross traffic, while

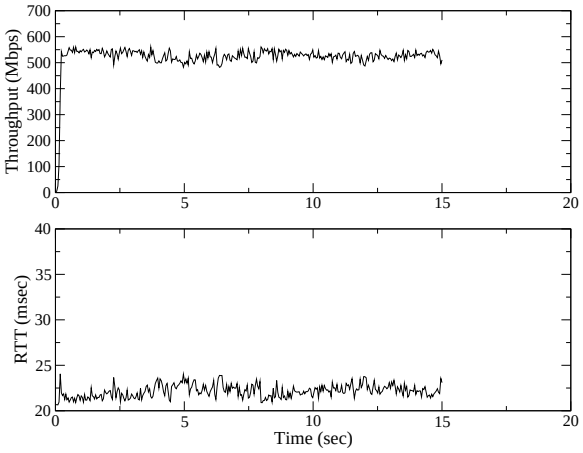

Fig. 11. With SOBAS: Gigabit path, unresponsive traffic (A=550Mbps).

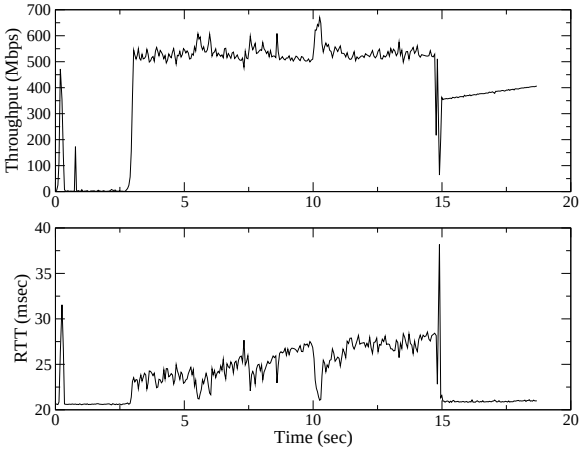

Fig. 12. Without SOBAS: Gigabit path, unresponsive traffic (A=550Mbps).

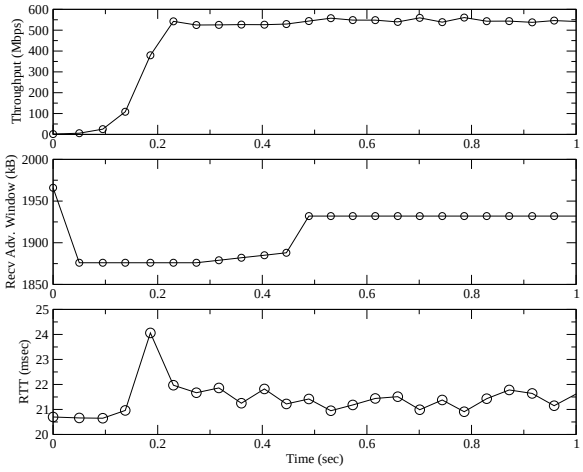

Fig. 13. Initial throughput, socket buffer size, and RTT with SOBAS in the path of Figure 11.

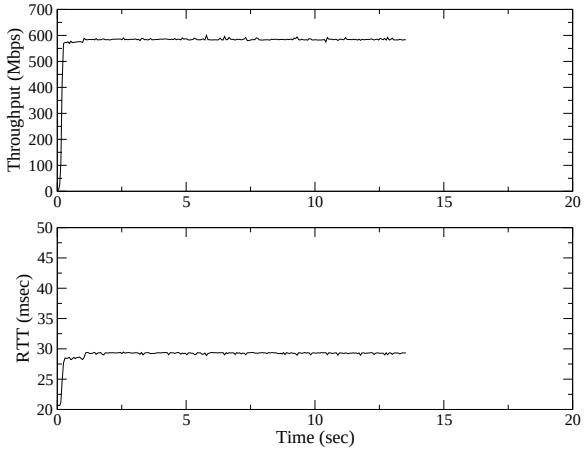

Fig. 14. With SOBAS: Gigabit path, buffer limited TCP traffic (A=450Mbps).

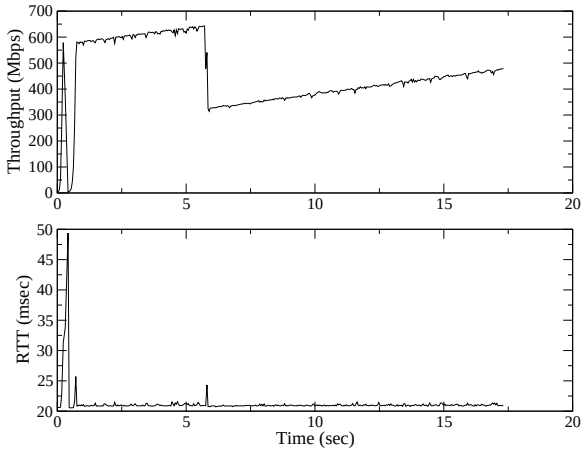

Fig. 15. Without SOBAS: Gigabit path, buffer limited TCP traffic (A=450Mbps).

the two other hosts (dual processor, 3GHz Intel Xeon, 2GB RAM, PCI bus 66MHz/64bit, Redhat 9) are the source and sink of the target transfer. We use NISTNet [28] to emulate an RTT of 20 msec in the path.

Figures 9 and 10 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 918Mbps in the former and 649Mbps in the latter. A trace analysis of the two connections shows that the non-SOBAS flow experienced multiple losses during the initial slow-start. After recovering from those losses, the transfer started a painfully slow congestion-avoidance phase at about 600Mbps, without ever reaching the available bandwidth of the path. SOBAS, on the other hand, avoided the slow-start losses using the packet-train based capacity estimate. Shortly afterwards, about 500ms after the transfer started, SOBAS detected the **flat-rate** condition and it set S to its final value. The RTT with SOBAS increased only by 3ms, from 20ms to 23ms.

We next consider the performance of SOBAS with congestion unresponsive cross traffic. Instead of generating random cross traffic, we use trace-driven cross traffic generation, "replaying" traffic from an OC-48 trace (IPLS-CLEV-20020814-093000-0), available at NLANR-MOAT [29]. The average rate of the cross traffic is 400Mbps. Notice that even though the packet sizes and interarrivals are based on real Internet traffic, this type of cross traffic does not react to congestion or increased RTTs. Figures 11 and 12 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 521Mbps in the former and 409Mbps in the latter. There are two loss events in the non-SOBAS flow. First, the initial slow-start caused major losses and several timeouts, which basically silenced the transfer for about 2.5 seconds. After the losses were recovered, the non-SOBAS flow kept increasing its window beyond the available bandwidth. As a result, the RTT increased to about 28ms, and then the transfer experienced even more losses followed by a slow congestion-avoidance phase.

SOBAS, on the other hand, determined successfully the point at which the available bandwidth was saturated, and it limited the socket buffer size before any losses occur. Figure 13 shows in more detail the initial phase of the SOBAS transfer. At the start of the transfer, SOBAS set S=1,875KB based on the initial capacity estimate. At about 0.2s, the transfer became buffer limited, and SOBAS started increasing linearly the socket buffer size. Shortly afterwards, at about 0.4s, the **flat-rate** condition was detected, and SOBAS set S to its final value. Notice that the RTT was increased by only 1-2ms.

Next, we evaluate SOBAS with congestion responsive TCP-based cross traffic. The latter is generated with a buffer limited IPerf transfer that cannot get more than 500Mbps due to its socket buffer size. Figures 14 and 15 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 574Mbps in the former and 452Mbps in the latter. Once more we observe that the non-SOBAS flow experienced losses at the initial slow-start, even though they were recovered quickly in this case. One more loss event occurred about 6 seconds after the start of the transfer, causing a major reduction in the transfer's throughput. SOBAS, on the other hand, detected the **flat-rate** condition shortly after the start of the transfer, avoiding any packet losses.

B. High-bandwidth paths

We next present similar results for two paths in which the available bandwidth varies between 400-600Mbps. These paths carry "live" Internet traffic, and so we cannot know the exact nature of the cross traffic.

The first path connects two different buildings at the Georgia Tech campus. The path is rate-limited to about 620Mbps at the IP layer. Because the RTT of this path is typically less

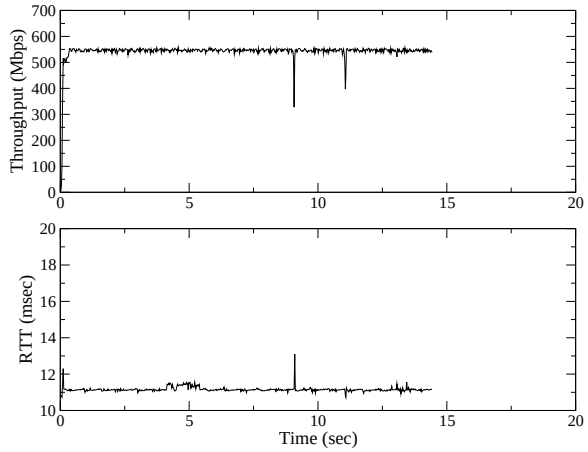

Fig. 16. With SOBAS: GaTech campus path (A=600Mbps).

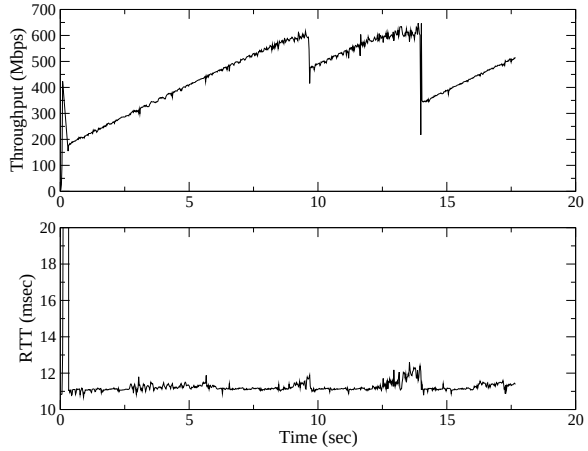

Fig. 17. Without SOBAS: GaTech campus path (A=600Mbps).

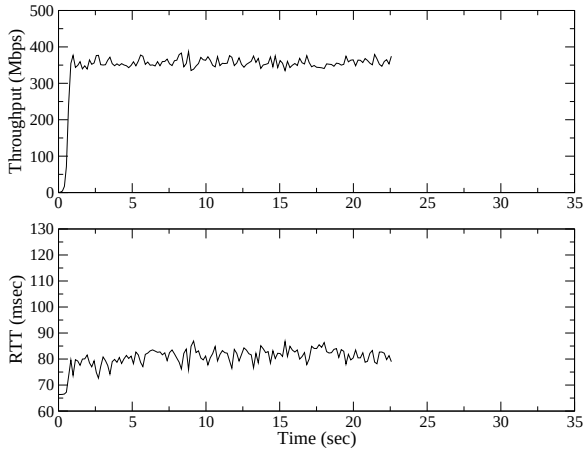

Fig. 18. With SOBAS: Path from GaTech to LBNL (A=450Mbps).

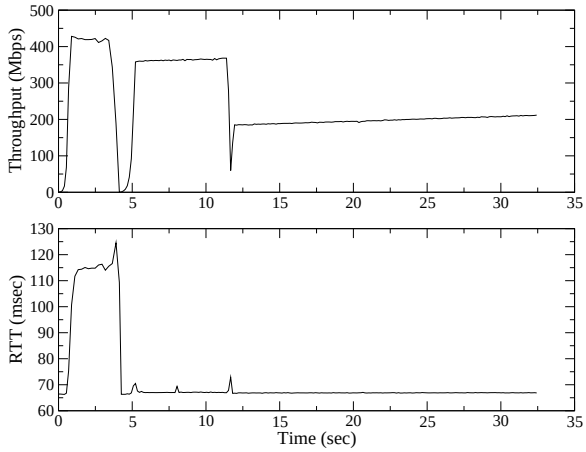

Fig. 19. Without SOBAS: Path from GaTech to LBNL (A=450Mbps).

than one millisecond, we again use NISTnet to create an additional delay of 10ms. Figures 16 and 17 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 542Mbps in the former and 445Mbps in the latter. Qualitatively, the results are similar to those of the Gigabit path without (or with unresponsive) cross traffic. Notice that the non-SOBAS flow pays a large throughput penalty due to the initial slow-start losses. The SOBAS flow avoids any losses, and it manages to get a constant throughput that is close to the capacity of the path.

A wide-area high-bandwidth network path that was available to us was the path from Georgia Tech to LBNL in Berkeley CA. The available bandwidth in the path is about 400Mbps, even though we do not know the location of the tight link. Figures 18 and 19 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 343Mbps in the former and 234Mbps in the latter. The large RTT in this path (T_e=60ms) makes the linear window increase during congestion-avoidance in the non-SOBAS flow to be ever slower than in the previous paths.

C. Typical paths

We finally show results from paths that provide less than 100Mbps of available bandwidth. Many paths between US and European universities and research centers fall into this class today.

The first path is from Georgia Tech to NYU. The path is limited by a Fast Ethernet, with a capacity of about 97Mbps at the IP layer. The available bandwidth that *pathload* measured was about 90Mbps. Figures 20 and 21 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 87Mbps in the former and 48Mbps in the latter. The non-SOBAS flow experienced several loss events, followed by slow recovery periods. Notice that the short throughput drops in the SOBAS flow are caused by RTT spikes (probably due to cross traffic bursts), and they do not correspond to loss events.

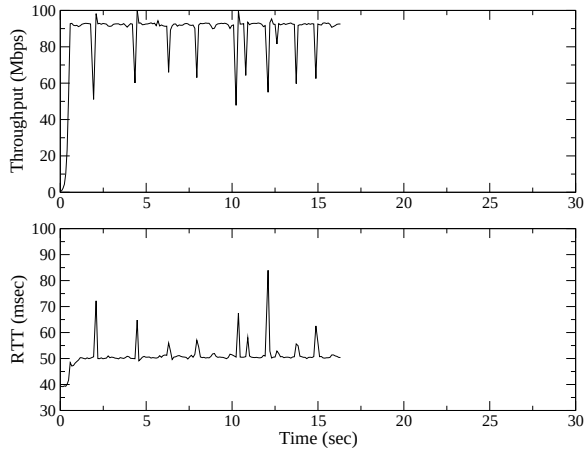

Fig. 20. With SOBAS: Path from GaTech to NYU (A=90Mbps).

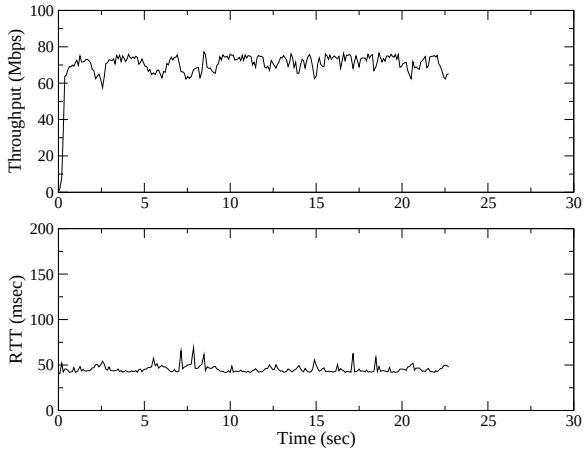

Fig. 22. With SOBAS: Path from GaTech to NYU (A=80Mbps).

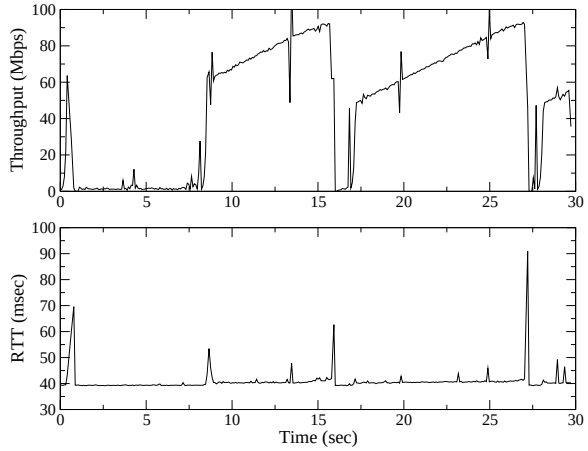

Fig. 21. Without SOBAS: Path from GaTech to NYU (A=90Mbps).

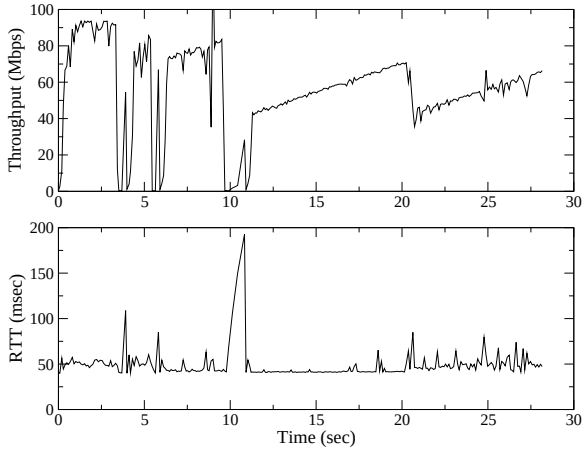

Fig. 23. Without SOBAS: Path from GaTech to NYU (A=80Mbps).

The next experiment was also performed at the GaTech-NYU path, but during a different time period. The available bandwidth in this case was about 80Mbps. Figures 22 and 23 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 70Mbps in the former and 57Mbps in the latter. An important point to take from these experiments is that SOBAS is robust to the presence of real Internet cross traffic, and it manages to avoid self-induced losses even though the RTT measurements show significant RTT spikes.

The final experiment was performed at a path from Georgia Tech to a host at Lulea in Sweden. The capacity and available bandwidth for this path was 97Mbps and 40Mbps, respectively. Figures 24 and 25 show the throughput and RTT for this path with and without SOBAS, respectively. The average throughput is 33Mbps in the former and 20Mbps in the latter. An interesting point about this experiment was that the SOBAS flow did not manage to avoid the self-induced losses at the initial slow start. This is because the initial capacity estimate (93.5Mbps) was much higher than the available bandwidth. The losses were recovered in about 5 seconds, and SOBAS detected a **flat-rate** condition at about t=10s. There were no losses after that point.

VI. Related work

A. Socket buffer sizing techniques

An auto-tuning technique that is based on active bandwidth estimation is the *Work Around Daemon (WAD)* [3]. 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 [30]. A similar approach is taken by the NLANR Auto-Tuning FTP implementation [31]. Similar socket buffer sizing guidelines are given in [2] and [13].

The first proposal for automatic TCP buffer tuning was [14]. 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 sys-

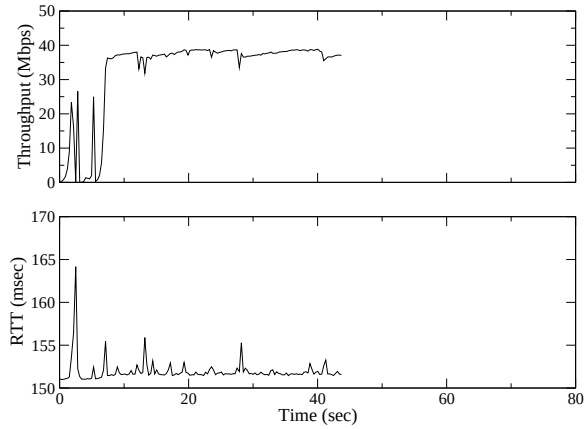

Fig. 24. With SOBAS: Path from GaTech to Lulea (A=40Mbps).

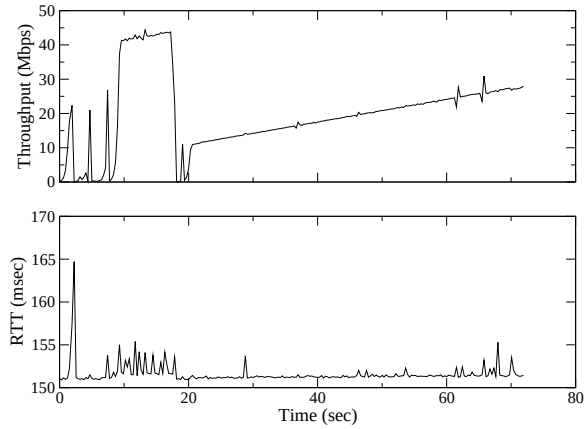

Fig. 25. Without SOBAS: Path from GaTech to Lulea (A=40Mbps).

tem. An important point about [14] is that the BDP of a path was estimated based on the congestion window (*cwnd*) of the TCP connection. 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 [15]. DRS measures the RTT of the path prior to the start of the connection. 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 six models in the previous section. A comparison of some socket buffer sizing mechanisms appears in [32].

We finally note that the 2.4 version of the Linux kernel sets the socket buffer size dynamically. 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. Also, Linux 2.4 adjusts the send socket buffer size dynamically, based on the available system memory and the transfer's send socket buffer backlog.

B. TCP congestion control modifications

Several researchers have proposed TCP modifications, mostly focusing on the congestion control algorithm, aiming to make TCP more effective in high-performance paths. Since our work focuses on techniques that require no changes in TCP, we do not review these proposals in detail here.

Floyd proposed High-Speed TCP [5], in which the window increase and decrease factors depend on the current congestion window. These factors are chosen so that a bulk TCP transfer can saturate even very high-bandwidth paths in lossy networks.

With similar objectives, Kelly proposed Scalable TCP [8]. An important difference is that Scalable TCP uses constant window increase and decrease factors, and a multiplicative increase rule when there is no congestion. With the latter modification, Scalable TCP recovers from losses much faster than TCP Reno.

TCP Westwood uses bandwidth estimation, derived from the dispersion of the transfer's ACKs, to set the congestion window after a loss event [10]. Westwood introduced the concept of "eligible rate", which is an estimate of the TCP fair share.

Another TCP variant that focuses on high-bandwidth paths is TCP FAST [6]. FAST has some important similarities with TCP Vegas[33]. The key idea is to limit the send window of the transfer when the RTTs start increasing. This is similar in principle with SOBAS, implying that FAST and Vegas also aim to saturate the available bandwidth in the path. An important difference is that SOBAS disables itself in congested paths, becoming as aggressive as a Reno connection. It is known, on the other hand, that Vegas is less aggressive than Reno in congested paths [34].

Recently, [35] proposed 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). Even though TCP-LP is not a socket buffer sizing scheme, it is similar to SOBAS in the sense that it aims to capture the available bandwidth. A major difference with TCP-LP is that SOBAS disables itself in congested paths, and so it would result in higher throughput than TCP-LP in such paths. Additionally, TCP-LP reduces the send window every time the RTTs show an increasing trend; this behavior would lead to lower throughput than SOBAS even in non-congested paths.

VII. Conclusions

Common socket buffer sizing practices, such as setting the socket buffer size to the default or maximum value, can lead to poor throughput. We developed SOBAS, an application-layer mechanism that automatically sets the socket buffer size while the transfer is in progress, without prior knowledge of any path characteristics. SOBAS manages to saturate the available bandwidth in the network path, without saturating the tight link buffer in the path. SOBAS can be integrated with bulk transfer applications, such as GridFTP, providing significantly better performance in non-congested wide-area network paths. We plan to integrate SOBAS with popular Grid data transfer applications in the future.

Acknowledgments

We are grateful to Steven Low (CalTech), Matt Mathis (PSC), Nagi Rao (ORNL), Matt Sanders (Georgia Tech), Brian Tierney (LBNL), Matt Zekauskas (Internet2) and the RON administrators for providing us with computer accounts at their sites. We also thank Karsten Schwan, Matt Wolf, Zhongtang Cai, Neil Bright and Greg Eisenhauer from Georgia Tech, and Qishi Wu from ORNL for the valuable comments and assistance. We also appreciate the availability of packet traces from the NLANR-PMA project, which is supported by the NSF cooperative agreements ANI-0129677 and ANI-9807479.

References

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

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

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

[4] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," in *Proceedings of ACM SIGCOMM*, pp. 263–274, Sept. 1999.

[5] S. Floyd, *HighSpeed TCP for Large Congestion Windows*, Dec. 2003. RFC 3649 (experimental).

[6] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proceedings of IEEE INFOCOM*, Mar. 2004.

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

[8] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," *ACM Computer Communication Review (CCR)*, Apr. 2003.

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

[10] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Using Adaptive Rate Estimation to Provide Enhanced and Robust Transport over Heterogeneous Networks," in *Proceedings IEEE ICNP*, Nov. 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] 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.

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

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

[15] 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.

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

[17] 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.

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

[19] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in *Proceedings of ACM SIGCOMM*, pp. 295–308, Aug. 2002.

[20] "Resilient Overlay Network (RON)." http://nms.lcs.mit.edu/ron/, June 2003.

[21] M. Jain, R. S. Prasad, and C. Dovrolis, "The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-Sizing," Tech. Rep. GIT-CERCS-03-02, Georgia Tech, Feb. 2003. Available at http://www.cercs.gatech.edu/tech-reports/.

[22] 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.

[23] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, pp. 197–211, Nov. 2001.

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

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

[26] S. Floyd, *Limited Slow-Start for TCP with Large Congestion Windows*, July 2003. Internet Draft: draft-ietf-tsvwg-slowstart-00.txt (work-in-progress).

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

[28] M. Carson and D. Santay, "NIST Net - A Linux-based Network Emulation Tool," *ACM Computer Communication Review*, vol. 33, pp. 111–126, July 2003.

[29] NLANR MOAT, "Passive Measurement and Analysis." http://pma.nlanr.net/PMA/, Dec. 2003.

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

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

[32] 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.

[33] L. S. Brakmo and L.L.Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas of Communications*, vol. 13, Oct. 1995.

[34] J. S. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment," in *Proceedings of ACM SIGCOMM*, pp. 185–195, 1995.

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

Appendix

We derive the Buffer Overflow Latency (BOL) D_o, i.e., the time period from the instant a TCP connection saturates a link to the instant that the link's buffer overflows for the

first time. The BOL is important because it determines the maximum time interval in which the SOBAS receiver should detect the **flat-rate** condition before losses occur.

Consider a TCP transfer with initial RTT T_0 limited by a link of capacity C and buffer B. Suppose that the transfer's throughput $R(t)$ reaches the capacity at time t_0, i.e., $R(t_0)$=C. Any following increase in the transfer's window is accumulated at the buffer and it results in increased RTT. Let $Q(t)$ be the backlog at the buffer at time $t \geq t_0$. The BOL is the minimum time period D_o such that $Q(t_0 + D_o) \geq B$.

The RTT $T(t)$ at time t is a function of the instantaneous backlog,

$$T(t) = T_0 + \frac{Q(t)}{C} \tag{5}$$

while the backlog $Q(t)$ is given by

$$Q(t) = W(t) - CT_0 = C[T(t) - T_0] \tag{6}$$

The previous equation shows that the backlog increase rate is equal to the window increase rate

$$\frac{dQ}{dt} = \frac{dW}{dt} \tag{7}$$

which depends on whether the transfer is in congestion-avoidance (CA) or slow-start (SS).

During congestion-avoidance, the window increases by one packet per RTT (ignoring delayed-ACKs for now). Thus

$$\frac{dW}{dt} = \frac{L}{T} = \frac{LC}{W} \tag{8}$$

From (7), we see that the BOL can be determined as follows

$$\int_{CT_0}^{CT_0+B} W dW = \int_0^{D_o} LC dt \tag{9}$$

Solving the previous equation gives us the BOL in congestion-avoidance:

$$D_o^{CA} = \frac{B^2 + 2BCT_0}{2LC} \tag{10}$$

Similarly, during slow-start the window increase rate is an entire window per RTT,

$$\frac{dW}{dt} = \frac{W}{T} = C \tag{11}$$

Therefore,

$$\int_{CT_0}^{CT_0+B} dW = \int_0^{RSL} C dt \tag{12}$$

which gives us the BOL in slow-start:

$$D_o^{SS} = \frac{B}{C} \tag{13}$$

In the presence of Delayed-ACKs, the window increase rate is reduced by a factor of two. In that case, Equations (10) and (13) should be replaced by

$$D_o^{CA} = \frac{B^2 + 2BCT_0}{LC} \tag{14}$$

and

$$D_o^{SS} = \frac{2B}{C} \tag{15}$$

respectively. Note that $D_o^{SS} << D_o^{CA}$.

The previous results show that the BOL is largely determined by the "buffer-to-capacity" ratio B/C, i.e., by the maximum queueing delay at the link. A common buffer provisioning rule is to provide enough buffering at a link so that the buffer-to-capacity ratio is equal to the maximum RTT among the TCP connections that traverse that link. For instance, a major router vendor recommends that B/C=500ms. In that case, (15) shows that SOBAS has at most one second, during slow-start, to detect that the transfer has saturated the available bandwidth, and to limit the socket buffer size.

Recall from §IV that SOBAS measures the received throughput every two RTTs, and that it detects the **flat-rate** condition after two successive constant measurements when it is in states (1) and (2) (see Figure 8). Thus, the minimum time period in which SOBAS can limit the socket buffer size is approximately four RTTs. In other words, SOBAS is effective in avoiding losses during slow-start as long as

$$4 \times T < D_o \tag{16}$$

For B/C=500ms, we have that D_o=1sec and SOBAS avoids losses as long as the RTT of the target transfer is $T <$ 250ms. For transfers with larger RTTs some losses may occur in slow-start. On the other hand, the BOL is significantly larger in congestion-avoidance, which explains why SOBAS is much more effective in avoiding losses during that phase.