AN EVOLUTIONARY APPROACH TO IMPROVE END-TO-END PERFORMANCE IN TCP/IP NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Ravi S. Prasad

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
April 2008
AN EVOLUTIONARY APPROACH TO IMPROVE END-TO-END PERFORMANCE IN TCP/IP NETWORKS

Approved by:

Dr. Constantine Dovrolis, Advisor
College of Computing
Georgia Institute of Technology

Dr. Mostafa Ammar
College of Computing
Georgia Institute of Technology

Dr. Ellen Zegura
College of Computing
Georgia Institute of Technology

Dr. Nick Feamster
College of Computing
Georgia Institute of Technology

Dr. Marina Thottan
Networking Research Lab
Bell-Labs, Alcatel-Lucent

Date Approved: 28 September 2007
To my parents,
The last six years have been a great learning experience for me. I owe most of it to my advisor Dr. Constantine Dovrolis. I started working with Constantine in Fall 2001 when I joined the Department of Computer Science at the University of Delaware. I will always be grateful to him for having confidence in me and taking me as his research advisee. He has been a great mentor and friend since then. Over the years, he has always been there to guide my research in the right direction. I greatly appreciate his patient support during tough times of my PhD career, and inspiration and encouragement during happy times. I feel really lucky to have Constantine as my advisor. He is the best.

I would like to thank my dissertation committee members Dr. Mostafa Ammar, Dr. Ellen Zegura, Dr. Nick Feamster and Dr. Marina Thottan. They have taken interest in my work and their valuable comments have helped greatly in improving my dissertation. Special thanks to Marina, who was my mentor during my internship at the Bell Labs in the summer of 2006 and kindly agreed to be on my dissertation committee. Working with her made those three months not only fruitful, but also enjoyable.

Everyone in the networking group at Georgia Tech made my stay here fun and memorable. I can’t imagine going through the ups and downs of graduate studies without their help and pleasant distractions. Big thanks to all of them. I greatly appreciate my friendship and intellectual discussions with Manish Jain, Amogh Dhamdhere, Sridhar Srinivasan, Shashidhar Merugu, Pradnya Karbhari, Hyewon Jun, Abhishek Kumar, Qi He, Ruomei Gao, Christos Gkantsidis and Richard Liston. Special thanks go to Manish who has been a great friend all these years. I will miss those long but untiring hours of arguments with him that helped shape my views on life, work and everything in between. I have learned a lot from him.

I am thankful to Shubhra Misra, Muthukumar Narayanswamy, Prabir Barooah and Antonios Danalis, for their friendship when I was in Delaware. Encouragements from Ravi
Kokku, Dhirendra Singh and Malay Kumar were instrumental in getting me started on graduate studies.

The most beautiful thing that happened to me during my tenure at Georgia Tech was meeting my wife, Yarong. All this time that I knew her, she was always there for me. Her support, patience and encouragement were vital to my PhD. I will always be grateful for her love and companionship.

Finally, I would like to thank my family in India for their continuous and unconditional support for all my endeavors. This thesis is dedicated to my parents who have endured tremendous hardships in sending me and all of my siblings on the right path. Without their sacrifices I would have been nowhere close to where I find myself today. I am so lucky to have grown up in the most loving family in the world. Rama bhaiya and Daya bhaiya are and will always be sources of inspiration for me. I am very grateful to Shambhu bhaiya, Sweety and Moni for their friendship. I would also like to thank newer members of my family, my sisters-in-law Shushma bhabhi and Babita bhabhi, and wonderful nieces and nephews Raj, Anjali and Aakash. Even though I was not there, you all were always there in my thoughts.
# TABLE OF CONTENTS

DEDICATION ................................................................. iii

ACKNOWLEDGEMENTS .................................................. iv

LIST OF TABLES ............................................................ ix

LIST OF FIGURES .......................................................... x

SUMMARY ................................................................. xiii

I INTRODUCTION ......................................................... 1

1.1 Contributions ....................................................... 4

1.1.1 Socket Buffer Sizing ............................................. 4

1.1.2 Router Buffer Provisioning ..................................... 5

1.1.3 Congestion Responsiveness of TCP-based Applications .... 5

1.2 Thesis Organization .................................................. 6

II RELATED WORK ...................................................... 7

2.1 TCP Congestion Control Modifications ........................... 8

2.2 Socket Buffer Sizing Techniques .................................. 9

2.3 Router Buffer Sizing ............................................... 10

2.4 Congestion Responsiveness and Flow Arrival Process ........ 12

III LOSS-BASED AND DELAY-BASED CONGESTION AVOIDANCE ....... 14

3.1 LCA in High Bandwidth-Delay Paths ............................. 15

3.2 Challenges for DCA .................................................. 17

3.2.1 Maximum Delay Variation and Network Buffer Size ....... 18

3.2.2 RTT Sampling Rate ............................................. 18

3.2.3 Degree of Multiplexing ........................................ 19

3.2.4 Presence of Random Losses ................................... 20

3.3 Summary ............................................................. 21

IV SOCKET BUFFER AUTO SIZING ..................................... 22

4.1 Socket Buffer Size and TCP Throughput ......................... 25

4.2 Maximum Feasible Throughput and Available Bandwidth .... 27
LIST OF TABLES

1  Simulation parameters ............................................. 73
2  CTR of the analyzed TCP traffic at various links. ................. 115
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time series of the throughput and the RTT of a TCP transfer between GT and NYU without socket buffer size limitation.</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Time series of the throughput and the RTT of a TCP transfer between GT and NYU with limited socket buffer size of 470KB.</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Throughput and RTT of a 200MB transfer with $S=128$KB.</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Throughput and RTT of a 200MB transfer with $S=400$KB.</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Throughput and RTT of a 200MB transfer with $S=550$KB.</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Throughput and RTT of a 200MB transfer with $S=900$KB (max).</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Throughput and RTT of a 30MB transfer in a congested path with $S=30$KB.</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Throughput and RTT of a 30MB transfer in a congested path with $S=S_\infty$.</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>Basic SOBAS algorithm</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>SOBAS state diagram</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>With SOBAS: Gigabit path, no cross traffic ($A=950$Mbps).</td>
<td>42</td>
</tr>
<tr>
<td>12</td>
<td>Without SOBAS: Gigabit path, no cross traffic ($A=950$Mbps).</td>
<td>42</td>
</tr>
<tr>
<td>13</td>
<td>With SOBAS: Gigabit path, unresponsive traffic ($A=550$Mbps).</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>Without SOBAS: Gigabit path, unresponsive traffic ($A=550$Mbps).</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>Initial throughput, socket buffer size, and RTT with SOBAS in the path of Figure 13.</td>
<td>44</td>
</tr>
<tr>
<td>16</td>
<td>With SOBAS: Gigabit path, buffer limited TCP traffic ($A=450$Mbps).</td>
<td>45</td>
</tr>
<tr>
<td>17</td>
<td>Without SOBAS: Gigabit path, buffer limited TCP traffic ($A=450$Mbps).</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>With SOBAS: GaTech campus path ($A=600$Mbps).</td>
<td>47</td>
</tr>
<tr>
<td>19</td>
<td>without SOBAS: GaTech campus path ($A=600$Mbps).</td>
<td>47</td>
</tr>
<tr>
<td>20</td>
<td>With SOBAS: Path from GaTech to LBNL ($A=450$Mbps).</td>
<td>48</td>
</tr>
<tr>
<td>21</td>
<td>Without SOBAS: Path from GaTech to LBNL ($A=450$Mbps).</td>
<td>48</td>
</tr>
<tr>
<td>22</td>
<td>With SOBAS: Path from GaTech to NYU ($A=90$Mbps).</td>
<td>50</td>
</tr>
<tr>
<td>23</td>
<td>Without SOBAS: Path from GaTech to NYU ($A=90$Mbps).</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>With SOBAS: Path from GaTech to NYU ($A=80$Mbps).</td>
<td>51</td>
</tr>
<tr>
<td>25</td>
<td>Without SOBAS: Path from GaTech to NYU ($A=80$Mbps).</td>
<td>51</td>
</tr>
<tr>
<td>26</td>
<td>With SOBAS: Path from GaTech to Lulea ($A=40$Mbps).</td>
<td>52</td>
</tr>
<tr>
<td>27</td>
<td>Without SOBAS: Path from GaTech to Lulea ($A=40$Mbps).</td>
<td>52</td>
</tr>
</tbody>
</table>
Schematic diagram of the experimental testbed. ........................................... 57
Link utilization as a function of the router buffer size for the experiments $U_{1000}, U_{1200}$ and $U_{3000}$. ............................................................... 60
Fraction of time a link is under heavy-load (i.e., more than 90% or 95% utilized) in different averaging time scales. The average utilization over the 4-minute duration of this experiment is 68%. ................................. 61
Median per-flow throughput as a function of the buffer size in the $U_{1000}$ experiments. ................................................................. 62
Median per-flow throughput as a function of the buffer size in the $U_{1200}$ experiments. ................................................................. 62
Median per-flow throughput of large flows as a function of the buffer size in the $U_{3000}$ experiments. ..................................................... 63
Median per-flow throughput of small flows as a function of the buffer size in the $U_{3000}$ experiments. ..................................................... 63
Average per-flow TCP throughput as a function of flow size for a small buffer size (30KB). ................................................................. 64
Average per-flow TCP throughput as a function of flow size for a medium buffer size (600KB). ................................................................. 65
Average per-flow TCP throughput as a function of flow size for a large buffer size (38MB). ................................................................. 66
Average TCP throughput as a function of the router buffer size when the loss probability and average queueing delay are given by the $M/M/1/B$ equations in the heavy-load regime ($\rho \approx 1$). ............................................. 70
Simulation set up .................................................................................. 72
Loss probability as a function of buffer size for low $\Gamma$ ($N_{in}=20, U=100, \Gamma=0.1$). 73
Queueing delay as a function of buffer size for low $\Gamma$ ($N_{in}=20, U=100, \Gamma=0.1$). 74
Loss probability as a function of buffer size for high $\Gamma$ ($N_{in}=1000, U=25, \Gamma=10$). ................................................................. 74
Queueing delay as a function of buffer size for high $\Gamma$ ($N_{in}=1000, U=25, \Gamma=10$). ................................................................. 74
TCP throughput for the S-model and L-model when the loss rate is given by the PL-form. ................................................................. 79
TCP throughput for the S-model and L-model when the loss rate is given by EX-form. ................................................................. 81
Sensitivity of the throughput to errors in the optimal buffer size, for the L-model (top) and the S-model (bottom). ............................................. 82
The fraction of bytes $f$ generated by flows that are active for the entire duration of a given time interval $T$ as a function of $T$. The error bars depict the minimum and the maximum values of the fraction $f$.

The flow arrival process is controlled by the session/application layer. Is that layer responsive to network congestion?

Simulation setup

The offered load with the TCP-OL and TCP-CL models (simulated, y-axis) as a function of the offered load $X$ that is predicted by the corresponding PS models (calculated, x-axis).

The offered load from the CL model tends to that of the OL model as $N$ and $T_i$ increase.

The loss rate as a function of the offered load for the TCP-OL and TCP-CL models.

The median and 90-th percentile of the queueing delay distribution as a function of the offered load for the TCP-OL and TCP-CL models.

Variance of the offered load with the TCP-OL and TCP-CL models for three averaging timescales.

Fraction of time the offered load is greater than the capacity for four averaging timescales.

The CDF of the length of consecutive overload periods for four averaging timescales. The average offered load is 95% of the capacity.

The CDF of the average number of active flows, measured at 1-sec intervals, from the TCP-OL and TCP-CL models when the offered load is 70% and 95% of the capacity.

The response of the traffic aggregate in the TCP-OL and TCP-CL models, when a congestion event is caused by a UDP stream of rate 15Mbps. The capacity is 50Mbps and the offered load (before the congestion event) is 47.5Mbps.

The time series of the number of active flows and of the queueing delay with the TCP-OL and TCP-CL models when a congestion event is caused by a CBR UDP stream.

The TCP feedback loop at the transport layer cannot avoid persistent overload if there is no session layer congestion control.

Packet interarrivals from two transfers within the same connection.

Timeline of two sessions. Each session consists of several transfers.

Classification of different sessions from the same user as open-loop or closed-loop.
Despite the persistent change and growth that characterizes the Internet, the Transmission Control Protocol (TCP) still dominates at the transport layer, carrying more than 90% of the global traffic. Despite its astonishing success, it has been observed that TCP can cause poor end-to-end performance, especially for large transfers and in network paths with high bandwidth-delay product. In this thesis, we focus on mechanisms that can address key problems in TCP performance, without any modification in the protocol itself. This evolutionary approach is important in practice, as the deployment of clean-slate transport protocols in the Internet has been proved to be extremely difficult. Specifically, we identify a number of TCP-related problems that can cause poor end-to-end performance. These problems include poorly dimensioned socket buffer sizes at the end-hosts, suboptimal buffer sizing at routers and switches, and congestion unresponsive TCP traffic aggregates. We propose solutions that can address these issues, without any modification to TCP.

In network paths with significant available bandwidth, increasing the TCP window till observing loss can result in much lower throughput than the path’s available bandwidth. We show that changes in TCP are not required to utilize all the available bandwidth, and propose the application-layer SOcket Buffer Auto-Sizing (SOBAS) mechanism to achieve this goal. SOBAS relies on run-time estimation of the round trip time (RTT) and receive rate, and limits its socket buffer size when the receive rate approaches the path’s available bandwidth. In a congested network, SOBAS does not limit its socket buffer size. Our experiment results show that SOBAS improves TCP throughput in uncongested network without hurting TCP performance in congested networks.

Improper router buffer sizing can also result in poor TCP throughput. Previous research in router buffer sizing focused on network performance metrics such as link utilization or loss rate. Instead, we focus on the impact of buffer sizing on end-to-end TCP performance. We find that the router buffer size that optimizes TCP throughput is largely determined by
the link’s output to input capacity ratio. If that ratio is larger than one, the loss rate drops exponentially with the buffer size and the optimal buffer size is close to zero. Otherwise, if the output to input capacity ratio is lower than one, the loss rate follows a power-law reduction with the buffer size and significant buffering is needed. The amount of buffering required in this case depends on whether most flows end in the slow-start phase or in the congestion avoidance phase.

TCP throughput also depends on whether the cross-traffic reduces its send rate upon congestion. We define this cross-traffic property as congestion responsiveness. Since the majority of Internet traffic uses TCP, which reduces its send rate upon congestion, an aggregate of many TCP flows is believed to be congestion responsive. Here, we show that the congestion responsiveness of aggregate traffic also depends on the flow arrival process. If the flow arrival process follows an open-loop model, then even if the traffic consists exclusively of TCP transfers, the aggregate traffic can still be unresponsive to congestion. TCP flows that arrive in the network in a closed-loop manner are always congestion responsive, on the other hand. We also propose a scheme to estimate the fraction of traffic that follows the closed-loop model in a given link, and give practical guidelines to increase that fraction with simple application-layer modifications.
The Internet has been constantly evolving ever since its inception as ARPAnet. It has grown from a network of a few hundreds of hosts to millions of hosts supporting an ever-increasing number and variety of applications and services. Despite all the changes in the Internet, the Transmission Control Protocol (TCP) remains the most widely used transport protocol, carrying more than 90% of the traffic [81]. TCP is used in a wide range of applications and network environments. For example, the range of applications that use TCP include delay-sensitive interactive applications, bandwidth-sensitive bulk transfers, and delay-jitter-sensitive multimedia applications. TCP is being used in both wired and wireless network paths, with bandwidth ranging from a few Kbps to several Gbps and propagation delays from a few milliseconds to several seconds.

While TCP is the primary mechanism in the Internet for resource management and sharing, it is not the best protocol for all the applications and environments that use TCP. For example, in a high bandwidth-delay path, a bulk transfer application can get significantly lower throughput than the available bandwidth of the path, due to halving and subsequent linear increase of TCP’s congestion window after a single packet drop. A measurement study showed that in the Internet2, 90% of the “bulk” TCP transfers (i.e., transfers of more than 10MB) receive throughput less than 5Mbps [89]. Some research works have tried to identify different causes that limit a TCP transfer’s throughput [78, 105].

Recently, several researchers have suggested replacing TCP with “clean-slate” transport protocols and congestion control designs. These proposals, however, have not been successful in terms of deployment, at least yet. Some possible reasons follow:

- Replacement costs of legacy infrastructures: Some of the proposed clean-slate solutions call for modifying and/or replacing existing networking infrastructure in terms of software and hardware. The cost of such changes can be prohibitively high. They are
especially harder to achieve given that the benefits of such changes have not yet been established.

- **Difficulties in finding a suitable replacement of TCP:** With many different alternatives available, it is not clear which one is the most appropriate replacement. This is mainly due to the failure of research studies to reach a generally acceptable conclusion. Furthermore, the lack of a realistic model for Internet traffic makes very large-scale experiments difficult. Hence, the performance of any new transport protocol can only be tested once it has been completely deployed, making this a vicious cycle.

- **Co-existence of different TCP replacements:** It is also not clear how these new protocols interact with each other if they would had to coexist. Most works evaluating TCP improvements assume that either only the new protocol exist, or a fraction of the flows use the new protocol while the rest still use current TCP.

- **Distributed administration and control:** The difficulty in deployment of a new protocol is compounded by the decentralized nature of the Internet’s administration today, where various autonomous entities control different parts of the network.

As the difficulty in deploying any changes depends on where they are required, we use the deployment location as the criterion to categorize different proposals. The simplest of the modifications are those requiring changes only at the application layer. Such changes are easiest to deploy as they do not require any changes in the network infrastructure or standards. The modifications that we propose in this thesis fall into this category.

The second category of modifications requires changes at the transport layer. These are more difficult to deploy than the application changes because they involve updates in standards and protocol specifications. In addition, such changes have to satisfy conflicting requirements of different applications. This class of modifications can be further divided in those requiring changes only at one end, the sender or the receiver. Examples for this include, NewReno [36], HISTCP [31], STCP [55] as they seek to modify the sender-side behavior of TCP. TCP-SACK [68] is an example of modifying protocol behavior at both ends.
Finally, some proposals may also require changes at the network layer. Initial work on seeking congestion feedback from the network can be traced back to ATM-ABR [76]. Recent proposals of seeking modifications at the network layer to enhance end-to-end performance include XCP [53] and RCP [26]. Unless all routers and switches in the path are updated, such changes either would not work or would not perform as intended.

These proposals can, furthermore, be classified as evolutionary and clean-slate approaches. In an evolutionary approach, improvements can be made by local changes and without requiring network-wide changes. They should also coexist with other approaches, existing or new, without being unfair\(^1\) to them. In a clean-slate approach, on the other hand, network-wide changes are acceptable or even required for the proposed scheme to function.

In this thesis, we take an evolutionary approach to address performance problems observed with TCP. While there exist many other proposals addressing these problems that can be classified as evolutionary, we further simplify our approach by not requiring any changes in the protocol specification or the implementation of TCP. Our solutions require minimal changes at the application/middleware layer. For example, our SOcket Buffer Auto-Sizing (SOBAS) scheme require changes only at the receiver side application. Solutions to improve congestion responsiveness can be implemented at both ends of an application. Our router buffer sizing recommendation takes place at the network layer, but the change itself is only related to the configuration of the router and it does not require network-wide changes. Furthermore, changes can be made at each router independently of each other.

We investigate possible causes of TCP’s poor performance and suggest solutions, instead of retiring TCP and starting out with a completely new protocol. We argue that the loss-based congestion control of TCP can be enhanced with delay-based congestion avoidance techniques, implemented at the application layer. We also identify a number of factors that are beyond TCP’s congestion control and lead to poor end-to-end TCP throughput and propose solutions that can alleviate their negative effects. These factors include poor buffer sizing at the end-hosts, suboptimal buffer size at routers and switches, and congestion

\(^{1}\)In this thesis, the term “fair” refers to “fairness in TCP-sense” unless otherwise specified.
unresponsive applications. The problems that we describe and address in this thesis can still be relevant to new transport protocols that may appear at the Internet in the future.

1.1 Contributions

In this thesis, we show that significant TCP throughput improvements can be obtained without changing the TCP protocol or its implementation. Specifically, we develop an application layer mechanism that improves TCP performance and requires modifications only at the receiver side of the application. We find a simple and important parameter for provisioning router buffer that maximizes average per-flow TCP throughput. We show the importance of a feedback loop in the session generation process for the Internet traffic and develop a methodology to measure congestion responsiveness of a traffic aggregate at a link.

In the following sections, we summarize the main contributions of this thesis.

1.1.1 Socket Buffer Sizing

TCP’s blind probing for more available bandwidth can lead to packet losses in an otherwise uncongested path. These losses are called *self-induced losses*. Such losses in turn can lead to reduced throughput for the TCP flow and lowered utilization for that path. In this work, we develop an application layer mechanism that avoids this lose-lose situation. This mechanism, called SOcket Buffer Auto-Sizing (SOBAS), prevents TCP from probing beyond the path’s available bandwidth by limiting its socket buffer size in an uncongested path. In congested paths, packet drops occur regardless of “probing” from any particular flow. Thus, there is no incentive for limiting one’s socket buffer size in such paths.

SOBAS uses receive rate and RTT estimation to determine if it is operating in a congested path or not and what should be the socket buffer size in the latter case. Our experiments in several high bandwidth paths show that SOBAS can provide a 20% to 80% higher throughput compared to TCP transfers that do not limit the socket buffer size. SOBAS improves TCP throughput in uncongested paths and allows TCP to obtain its fair-share in congested paths.
1.1.2 Router Buffer Provisioning

Previous research in router buffer sizing focused on network performance metrics such as link utilization or loss rate [9, 23, 73, 74, 97]. These works often use open-loop traffic or only analyze persistent TCP flows. In this work, we focus on the impact of buffer sizing on end-to-end TCP performance, while considering the more realistic case of non-persistent TCP flows with a heavy-tailed size distribution. We find that the router buffer size that optimizes TCP throughput is largely determined by the link’s output to input capacity ratio. If that ratio is larger than one, the loss rate drops exponentially with the buffer size and the optimal buffer size is close to zero. Otherwise, if the output to input capacity ratio is less than one, the loss rate follows a power-law reduction with the buffer size and significant buffering is necessary. The amount of buffering required in this case depends on whether most flows end in the slow-start phase or in the congestion avoidance phase [82].

1.1.3 Congestion Responsiveness of TCP-based Applications

The throughput of a TCP flow also depends on whether the cross-traffic reduces its send rate upon congestion. This cross-traffic property is called its congestion responsiveness. The conventional wisdom is that if individual transfers react to congestion the way TCP does, then the traffic aggregate at a link will also be congestion responsive. Consequently, there have been proposals of applying similar congestion control algorithms in non-TCP protocols (e.g., TCP-Friendly control [35] or TCP tunnels [63]). This results from using persistent TCP connections, i.e., transfers that have infinite data to send and that last indefinitely, as the main network traffic model. When we view all TCP flows as non-persistent (i.e., with a finite size and duration), then the key issue is the random process that determines the arrival of flows, rather than packets, into the network. We examine two flow generation models. First, a closed-loop (CL) model where each user from a certain population can generate a new flow only after the completion of her previous session. Second, an open-loop (OL) model where flows arrive independently of previous flows [80]. These two models produce traffic with very different traffic aggregates, even if each flow is controlled by TCP. We show that the closed-loop model produces congestion responsive traffic, while the open-loop
model results in unresponsive traffic.

We have also analyzed several traffic traces collected at a dozen of Internet links in order to estimate the fraction of traffic that can be mapped to either the OL or CL model. Our measurements show that about 60-80% of traffic to/from well-known ports follow the CL model [81].

1.2 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives a background and discusses work related to the topics covered in this thesis. Chapter 3 gives the pros and cons of Loss-based Congestion Avoidance (LCA) and Delay-based Congestion Avoidance (DCA). In Chapter 4, we present the application layer mechanism SOBAS that improves TCP performance in uncongested paths without adversely affecting it in congested paths. The problem of selecting a router buffer size to optimize TCP performance is discussed in Chapter 5. Chapter 6 discusses the congestion responsiveness of traffic aggregates and Chapter 7 presents a methodology to measure congestion responsiveness at an Internet link. Finally, Chapter 8 summarizes the contributions of this thesis and outlines some future directions.
CHAPTER II

RELATED WORK

This chapter summarizes past work in the areas related to the subject of this thesis. These works include suggestions for modifying/replacing TCP, socket buffer sizing, and router buffer sizing. We also discuss works related to the open-loop and the closed-loop modeling and to the congestion responsiveness and the stability of the Internet traffic.

Proposals for modifying TCP vary from tuning/changing AIMD parameters of TCP to replacing the congestion control mechanism and requiring cooperation from each router in the path. Most of the previous work on modifying socket buffer size attempt to set the socket buffer size to some notion of bandwidth-delay product. They differ in their interpretation and estimation of bandwidth and delay. These works try to remove the arbitrarily fixed socket buffer size provided for all connections by the default value from the operating system. They, however, do not try to infer the network condition, relating it to the socket buffer size that will be most favorable for a given connection.

Past work on the router buffer provisioning have focused on network-centric performance matrices such as the link utilization and the packet drop rate. Specifically, they attempt to answer the question, "What is the minimum buffer size at a given router that would maximize link utilization while keeping the loss rate below an acceptable level?", without considering its impact on end-to-end performance.

Most of past work in the area of congestion responsiveness treat all TCP flows as responsive traffic and suggest ways to make non-TCP traffic responsive using TCP tunneling, TCP-friendly rate limiting etc. Such work fails to capture that even an aggregate of TCP flows can be unresponsive to congestion if the flow arrival process follows an open-loop model, as we show in this thesis.
2.1 TCP Congestion Control Modifications

Several research works have proposed TCP modifications, mostly focusing on the congestion control algorithm, aiming to make TCP more effective in high-bandwidth paths. As our work does not require changes in TCP, we do not discuss these works in detail.

Floyd proposed High-Speed TCP [31], 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 [55]. 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. Xu et al. proposed BIC-TCP that performs a binary search for a suitable congestion window size [103].

Some proposed TCP replacement protocols require cooperation from routers in the path. Katabi et al. proposed a new transport protocol eXplicit Control Protocol (XCP) [53]. XCP requires each router to notify XCP sources the amount of bandwidth available to them. Dukkipati et al. proposed Rate Control Protocol (RCS) where every router provides a constant rate to all of the active flows [26]. This attempts to get a more closer emulation of processor sharing among Internet flows.

TCP Westwood uses bandwidth estimation, derived from the dispersion of the transfer’s ACKs, to set the congestion window after a loss event [99]. 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 [51]. FAST has some important similarities with TCP Vegas [18]. 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 [2].

Recently, Kuzmanovic et al. [61] 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 only the available bandwidth. A major difference with TCP-LP is that SOBAS disables itself in congested paths with little or no available bandwidth, 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.

2.2 Socket Buffer Sizing Techniques

An auto-tuning technique that is based on active bandwidth estimation is the Work Around Daemon (WAD) [27]. 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 [40]. A similar approach is taken by the NLANR Auto-Tuning FTP implementation [65]. Similar socket buffer sizing guidelines are given in [94] and [69].

Automatic TCP buffer tuning was first proposed by Semke et al. [88]. The goal of that work was to allow a host (typically a server) to fairly share kernel memory between multiple ongoing connections. An important point about this work is that the bandwidth-delay-product (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 [39]. 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. Weigle et al. present a comparison of some socket buffer sizing
mechanisms [101].

Recent versions of the Linux kernel set 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 adjusts the send socket buffer size dynamically, based on the available system memory and the transfer’s send socket buffer backlog.

2.3 Router Buffer Sizing

Most of the work on buffer provisioning for routers and switches have focused the problem of high losses and capacity underutilization. As the size of router buffer increases, the loss rate decreases and the utilization increases. Such works provide a minimum buffer size that will produce an acceptable loss rate and utilization.

Several queueing theoretic papers analyze either the loss probability in finite buffers, or the tail probability in infinite buffers. Usually, however, that modeling approach considers exogenous (or open-loop) traffic models, in which the packet arrival process does not depend on the state of the queue. For instance, the paper by Kim and Shroff models the input traffic as a general Gaussian process, and derives an approximate expression for the loss probability in a finite buffer system [57].

An early experimental study by Villamizar and Song [97] recommends that the buffer size should be equal to the Bandwidth-Delay Product (BDP) of that link. The “delay” here refers to the RTT of a single and persistent TCP flow that attempts to saturate that link, while the “bandwidth” term refers to the capacity $C$ of the link. No recommendations are given, however, for the more realistic case of multiple TCP flows with different RTTs.

Appenzeller et al. [9] conclude that the buffer requirement at a link decreases with the square root of the number $N$ of “large” TCP flows that go through that link. According to their analysis, the buffer requirement to achieve almost full utilization is $B = (CT_{avg})/\sqrt{N}$, where $T_{avg}$ is the average RTT of the $N$ (persistent) competing connections. The key insight behind this model is that, when the number of competing flows is sufficiently large, which is usually the case in core links, the $N$ flows can be considered independent and
non-synchronized, and so the standard deviation of the aggregate offered load (and of the queue occupancy) decreases with the square root of $N$. An important point about this model is that it aims to keep the utilization close to 100%, without considering the resulting loss probability.

Morris was the first to consider the loss probability in the buffer sizing problem [73, 74]. That work recognizes that the loss rate increases with the square of the number of competing TCP flows, and that buffering based on the BDP rule can cause frequent TCP timeouts and unacceptable variations in the throughput of competing transfers [73]. That work also proposes the Flow-Proportional Queueing (FPQ) mechanism, as a variation of RED, which adjusts the amount of buffering proportionally to the number of TCP flows.

Dhamdhere and Dovrolis consider the buffer requirement of a Drop-Tail queue given constraints on the minimum utilization, maximum loss-rate, and when feasible, maximum queueing delay [23]. They derive the minimum buffer size required to keep the link fully utilized by a set of $N$ heterogeneous TCP flows, while keeping the loss rate and queueing delay bounded. However, the analysis of that paper is also limited by the assumption of persistent connections.

Enachescu et al. show that if the TCP sources are paced and have a bounded maximum window size, then a high link utilization (say 80%) can be achieved even with a buffer of a dozen packets [28]. The authors note that pacing may not be necessary when the access links are much slower than the core network links. It is also interesting that their buffer sizing result is independent of the path BDP.

Recently, the ACM Computer Communication Review has hosted a debate on buffer sizing through a sequence of letters. Dhamdhere and Dovrolis argue that the recent proposals for much smaller buffer sizes can cause significant losses and performance degradation at the application layer [24]. Similar concerns are raised by Vu-Brugier et al. in [98]. That letter also reports measurements from an operational link in which the buffer size was significantly reduced. Ganjali and McKeown discuss three recent buffer sizing proposals [9, 23, 28] and argue that all these results may be applicable in different parts of the network, as they depend on various assumptions and they have different objectives [38].
2.4 Congestion Responsiveness and Flow Arrival Process

The notion of congestion responsiveness is related to the issue of TCP-friendliness [35]. The latter, however, focuses on non-TCP traffic. Floyd and Fall [34] and Le et al. [62] proposed to control high-bandwidth flows to prevent persistent overload. Similarly, the Network Border Patrol [3] and ERUF [85] are mechanisms to limit transfers that receive higher throughput than the TCP-friendly rate. The latter is the average throughput that a persistent TCP connection would experience at the same path [34]. Zhao et al. [108] proposed a method to estimate the fraction of congestion unresponsive traffic at a link. In this work, we show that a traffic aggregate can be congestion unresponsive even if all the constituent flows are TCP or TCP-friendly flows.

Some previous works use non-persistent flow models, but often without discussing whether the OL or the CL model is more appropriate. Ben Fredj et al. [37] considered the OL model. They noted that the only reduction in the offered load upon a congestion event is due to aborted transfers. Such transfers, however, result in wasted throughput and user dissatisfaction. For this reason, the authors proposed admission control as the only efficient way to prevent persistent overload. Veciana et al. [96] considered the OL model and concluded that the Internet traffic may become unstable under certain conditions.

Over the last few years, and especially after the seminal work by Kelly et al. [54], several researchers applied control theory to examine the stability of TCP congestion control [60, 107, 66]. A key point about that line of work is that it assumes persistent TCP flows, and it focuses on the asymptotic stability of the queue size at the network bottleneck. The assumption of persistent flows removes from the problem the importance of the flow arrival process.

Heyman et al. [43] used a CL model to analyze the performance of Web-like traffic over TCP. They showed that the session goodput and the fraction of time the system has a given number of active sessions are insensitive to the distribution of session sizes and “think times”, and they only depend on the mean of these distributions. Berger and Kogan [11], as well as Bonald et al. [14], used a similar CL model to design bandwidth provisioning rules for meeting certain throughput-related QoS objectives.
Most of the previous work with the OL or the CL models assume that TCP congestion control can share the capacity of a link as a fluid PS server [86]. Kherani and Kumar [56] showed that the PS model is not always accurate, mostly because TCP transfers do not manage to keep the link fully utilized under certain conditions. In this work, we use the PS model just to gain some insight in the congestion responsiveness of the OL and the CL models. Our simulations, on the other hand, use actual TCP transfers.

Bondi and Whitt [15] examine the differences between the OL and CL models in the context of networks of queues, focusing on the relation between the average queue size at the bottleneck queue and the variability in the job service-time distribution.

In a paper that is closely related to our work, Schroeder et al. [87] compare the OL and CL models in a general context of job arrivals at a server. They highlight the differences between the two models in terms of the mean job completion time, and they focus on the effectiveness of different job scheduling policies with each model.

The open-loop and the closed-loop models are simple models for user and application behavior. Understanding such behavior, as well as their impact on network traffic, has been the subject of some previous work [21, 30, 91, 95]. These papers focus on HTTP, mostly because the application characteristics of Web browsing are well understood. Casilari et al. [21] present models for different levels of HTTP analysis, namely at the packet, connection, page or session levels. Smith et al. attempt to reconstruct HTTP sessions from unidirectional TCP header traces [91]. Feldmann [30] correlates bidirectional IP traces with HTTP header traces. Tran et al. [95] use payload information to identify related HTTP requests/responses and to study the effect of congestion on user behavior in terms of the probability of aborted and retried sessions.
CHAPTER III

LOSS-BASED AND DELAY-BASED CONGESTION AVOIDANCE

One fundamental requirement for a stable network is that its traffic reacts to congestion. The detection of congestion in the Internet, however, is difficult due to the fact that the network does not provide any congestion notification to traffic sources. There have been several proposals for adding explicit congestion notification (ECN) in the Internet [19, 33, 84]. ECN, however, has seen minimal deployment and has yet to become a main stream mechanism. In the absence of ECN, Internet flows need to detect congestion before they can respond to it.

End-systems can detect network congestion by either measuring increasing delays or packet losses in the path. Delay-based Congestion Avoidance (DCA) takes increasing delays as a sign of congestion, while Loss-based Congestion Avoidance (LCA) reacts to congestion only after experiencing a packet loss. The first algorithms based on LCA or DCA were proposed around the same time [44, 48] and the debate as to which congestion avoidance scheme is better still continues.

LCA schemes treat only packet losses as an indication of network congestion. Therefore, an LCA-based TCP transfer decreases its send-window to reduce the load in the network, only after experiencing a packet loss. The canonical example of LCA is Jacobson’s original TCP congestion control [44]. LCA as proposed in [7] is the most prevalent congestion control scheme in current Internet. In the paths with significant available bandwidth, however, LCA can be less than efficient. This is because an LCA sender keeps increasing its send-window, even after it has taken up the available bandwidth, eventually causing buffer overflow. The multiplicative decrease of the send-window after packet losses, and the subsequent additive increase, result in link under-utilization for significant time intervals. These “self-induced” packet drops not only result in an increased loss rate, but also cause decreased throughput and, at least with Drop-Tail queues, significant delay variations.
DCA schemes, on the other hand, attempt to control the send-window of a TCP transfer based on Round-Trip Time (RTT) measurements. The basic intuition is that, if the send-window is large enough to saturate the available bandwidth, then any further increase in the send-window will cause a queue build-up at the tight link and thus an increasing RTT. Consequently, a DCA-based TCP sender decreases the transfer’s send-window after it detects increasing RTTs. There are several variations of DCA schemes. Starting with Jain’s initial proposal in 1989 [48] and Mitra’s fundamental work [72], the networking research community looked into various ways of modifying DCA-based TCP, including TCP TriS [100], TCP Vegas [18], TCP BFA [10], and most recently TCP FAST [50]. Interest in DCA algorithms has re-emerged in the last couple of years with the emergence of high-speed networks, partly due to the fact that LCA-based TCP cannot efficiently utilize such high-bandwidth and long-distance network paths. Recently, however, measurement studies have shown that there is little correlation between increased delays (or RTTs) and congestive losses [8, 12, 67]. This experimental observation raises some serious doubts on whether DCA algorithms would be effective in practice, as their main assumption is that RTT measurements can be used to predict, detect, and avoid network congestion [42].

In this chapter, we highlight the performance disadvantages of LCA in high-speed path using Internet experiments and show that DCA can help mitigate this problem. We also discuss the challenges that a purely DCA-based TCP faces in more widespread use. We suggest some possible explanations for the weak correlations between delays and congestive losses observed in past studies, and identify conditions under which DCA schemes can fail to provide robust congestion control. Based on these observations, we present a socket buffer auto-sizing (SOBAS) mechanism in Chapter 4, which performs DCA at the application layer if this can help improve TCP’s performance, and falls back to LCA-based TCP in other cases.

### 3.1 LCA in High Bandwidth-Delay Paths

In this section, we show that LCA-based TCP’s reliance on packet drops for congestion detection can in fact cause such drops in an otherwise uncongested link. In paths with high
available bandwidth and sufficient router buffer size, DCA could prove to be an attractive alternative to avoid self-induced losses as caused by LCA.

We performed an analysis of some TCP transfers between a host at Georgia Institute Technology (GT) and a RON [1] node at New York University (NYU). This path had a FastEthernet link as the narrow and tight link, and had very little cross-traffic during most of our experiments. During these experiments, the path’s available bandwidth as measured with Pathload [45] was consistently about 94Mbps, the IP layer capacity of a FastEthernet link [83].

Figure 1 shows the time series of the TCP throughput and the RTT for the path between GT and NYU. The TCP throughput is measured as the application layer goodput and the RTT is measured by periodic ping messages. We observe that whenever the TCP throughput gets close to the path available bandwidth of 94Mbps, there is an increasing trend in the path RTT. As the LCA-based TCP ignores these RTT signals, it experiences packet drops following the increased RTT. The average throughput for this transfer remains only around 64Mbps, 30% less than the available bandwidth of the path.

Figure 1: Time series of the throughput and the RTT of a TCP transfer between GT and NYU without socket buffer size limitation.

Figure 2 shows the time series of the TCP throughput and the RTT for the same path. In this experiment run, however, we have limited the socket buffer size of the transfer to
the bandwidth-delay product of the path, thus making it behave like a DCA-based TCP. With the available bandwidth of 94Mbps and the RTT of 40msec, the transfer with a socket buffer size of 470KB does not attempt to increase its send rate beyond the path available bandwidth and thus keeps the RTT close to the path propagation delay of about 40msec. In this case, TCP does not experience any increased RTT, packet loss and the average throughput for the transfer remains close to the available bandwidth of the path.

![Graph showing throughput and RTT over time](image)

**Figure 2:** Time series of the throughput and the RTT of a TCP transfer between GT and NYU with limited socket buffer size of 470KB.

To summarize, LCA-based TCP can cause self-induced losses. Such losses result in poor TCP performance. DCA, on the other hand, avoids self-induced losses.

### 3.2 Challenges for DCA

In this section, we identify different factors that affect DCA-based TCP’s performance. These factors contribute to difficulties in accurate and timely measurement of increasing delays, weak correlations between increasing RTT and congestion events, and call for the need of LCA as a back-up mechanism to deal with packet drops that are uncorrelated to increasing RTT. We use the analysis below in the design of SOBAS, described in Chapter 4, which enforces DCA at the application layer, if deemed useful, with LCA at the transport layer.
3.2.1 Maximum Delay Variation and Network Buffer Size

In the following, we discuss whether the increase in the queuing delay is large with respect to the noise in the RTT measurement of a path. It also raises the importance of network buffer size as that determines the maximum increase in queuing delay.

Suppose that the network path of a TCP transfer has a minimum RTT $T_{\text{min}}$. $T_{\text{min}}$ is determined by the propagation and transmission delays along the path and it does not include any queuing delays. In addition, suppose that the tight link in the path has capacity $C_t$ and buffer size $B_t$. The maximum queueing delay in that link is $B_t/C_t$. If the queueing occurs only at the tight link, then the maximum RTT will be given by,

$$T_{\text{max}} = T_{\text{min}} + \frac{B_t}{C_t}.$$  \hspace{1cm} (1)

DCA algorithms control the send-window based on the measured RTT variation $\Delta T$, which will be less than the maximum queueing delay $B_t/C_t$. The first issue here is that, if $B_t/C_t \ll T_{\text{min}}$, then it may be difficult to robustly measure an increase in the RTT due to queueing at the tight link. Secondly, random noise in the RTT measurements, which is unavoidable in practice, or errors due to limited RTT timestamping resolution, can be comparable or even larger than the queueing delays in the tight link. In that case also, a DCA-based sender may not be able to detect the increased queueing delays at the tight link and to avoid congestive losses.

3.2.2 RTT Sampling Rate

In DCA algorithms, the sender measures the RTT using the transfer’s data packets. Even if every data packet generates an RTT measurement, the sender will effectively sample the path’s RTT every $T_s = L/R$ seconds, where $L$ is the data packet size and $R$ is the transfer’s send-rate over that time period. The RTT sampling rate will then be $R/L$. From Nyquist’s theorem, however, it is known that if the sampling rate is less than twice the maximum frequency in the spectrum of the sampled signal, we cannot reconstruct that signal. Even if we do not aim for a perfect signal reconstruction, the sampling rate should be at least comparable to the maximum signal frequency; otherwise, we are at risk of just sampling noise.
However, what determines the maximum frequency in the spectrum of the RTT signal? The RTT in a network path varies mostly due to queueing delays, and these delays are caused by variations in the incoming rate of the corresponding queues. To illustrate this point, let’s consider the queue of the tight link at a time instant $t_0$. Suppose that just prior to $t_0$ the queue had some available bandwidth $A$ and that the rate of our transfer was $R$. At $t_0$ a new cross traffic flow arrives at the tight link with rate $R_c > A$. This will cause a queue build-up at the tight link with rate $R_c - A$. If the tight link buffer was empty at $t_0$ and it has a maximum size of $B_t$ bytes, it will take $\frac{B_t}{R_c - A}$ seconds for that buffer to overflow.

In order for a DCA transfer to detect this RTT increase on time, it should have a much lower sampling period, i.e.,

$$\frac{L}{R} \ll \frac{B_t}{R_c - A} \quad (2)$$

or, equivalently, its rate should be significantly higher than the rate with which the backlog increases at the tight link, normalized by the buffer size at that link,

$$R \gg \frac{R_c - A}{B_t/L} \quad (3)$$

Equation (3) shows that if our transfer is a low throughput flow in a high-bandwidth path (i.e., $R \ll A$), it may be unable to accurately sample the RTT variations in the path. High-bandwidth cross traffic flows will be causing significant RTT variations which our transfer will not be able to detect, and therefore respond to, on time. This conjecture agrees with the experimental and simulation results of [12] and [67], according to which DCA algorithms often react incorrectly to RTT variations, especially in high-bandwidth paths.

3.2.3 Degree of Multiplexing

DCA is often described and modeled considering a single transfer using the tight link of the path. Even if cross traffic is part of the model, it is often assumed to not cause any queueing delay variations. In that context, any variations in a transfer’s RTT are caused by corresponding variations in that transfer’s send-window. It is then relatively simple to design DCA algorithms that adjust the send-window based on RTT measurements, so that the TCP transfer fully utilizes the path without causing congestion.
However, the situation becomes very different when we consider that the path carries multiple TCP transfers, and that they all dynamically adjust their windows based on their own perception of the network state. Specifically, consider that at some time instant our transfer’s send-rate is \( R \), while the capacity of the tight link is \( C_t \), fully utilized by several cross traffic TCP connections with aggregate rate \( R_c = C_t - R \). If \( R \gg R_c \), our transfer will have a major impact on the tight link’s queue, and so there will be a strong positive correlation between the window of our transfer and the measured RTTs. In such a case, we can expect a DCA algorithm to perform well.

If, however, \( R \ll R_c \), our transfer would be a minor contributor to the queueing delays at the tight link, and so the measured RTTs would be weakly correlated, or even uncorrelated, with that transfer’s window. Therefore, it is likely that our transfer will be reacting to RTT variations erroneously, mostly driven by the effect of cross traffic on the tight link, rather than by the effect of its own load. This is exactly what has been observed in [12] and [67] that DCA algorithms perform well in low-bandwidth tight links because there is typically only one connection using those links at a time. In high-bandwidth links, on the other hand, where there exist a large number of concurrent connections from different users, DCA schemes perform much worse. In such a case, DCA schemes cannot predict congestion accurately and often vary their send-windows erroneously [67].

### 3.2.4 Presence of Random Losses

Even though a DCA-based transport protocol would be primarily reacting to RTT variations, it also needs to react to packet losses by decreasing the send-window. Packet losses in network paths are unavoidable for two reasons. First, random packet drops occur independently of congestion due to bit errors, hardware/software errors, etc. Without global deployment of the Explicit Congestion Notification (ECN) bit, end hosts are not able to distinguish between congestive and random losses and therefore they need to react to both. Second, congestive packet losses can be due to the cross traffic, independent of our transfer’s send-window. Earlier in Figure 1, there were such packet drops at 17.5 and 40.0 sec, which are not preceded by path saturation and increase in queueing delay, and could have affected
DCA-based TCP as well. A TCP connection, however, has no means to infer whether an observed packet loss was caused because another flow needed to decrease its window as congestive losses can affect any of the active flows. To summarize, DCA flows can be affected by packet losses and they need to react to them by decreasing their windows.

How should a DCA flow react to a packet loss? This is mostly a fairness issue. If DCA schemes do not decrease their windows by the same factor that LCA schemes do, significant unfairness in the bandwidth distribution can occur. We believe that it is necessary that any proposed DCA protocol specifies in detail how it reacts to packet drops, and how it maintains (or not maintains) fairness to existing TCP congestion control algorithms.

Another way to look at this issue is that, even though DCA algorithms are able to avoid self-induced packet losses, and thus to achieve higher throughput in cases where all the losses are self-induced, DCA algorithms may not provide any performance benefit if they operate in network paths where random, or congestive losses due to cross traffic, are common. So, it is important that the evaluation of DCA schemes is also performed in lossy networks, as well as in networks where DCA flows compete with LCA flows.

3.3 Summary

We saw here that self-induced losses with LCA can result in significant performance degradation for TCP in high-speed networks. While DCA can successfully avoid such losses, there are several scenarios under which the use of increasing delays to detect congestion can be problematic. Depending on the traffic and network conditions, delay signals can have high variability, its increasing trend may last for a very small time, or may not be related to the send-window of any individual transfer. The analysis in this chapter provided some insight into shortcomings of both DCA and LCA. We use these insights in the design of an application layer mechanism, described in the next chapter, where we use both LCA and DCA in a complimentary fashion. Specifically, we use DCA when LCA would cause self-induced losses and use LCA when the path is lossy independent of our connection’s send-window.
CHAPTER IV

SOCKET BUFFER AUTO SIZING

The emergence of the high-speed network 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 laboratories, 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 such users expect great 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 throughput less than 5Mbps [89].

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) [94]. 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 [27, 5].
Researchers have focused on these problems, pursuing mostly three approaches: TCP modifications [31, 51, 53, 55, 59, 99], parallel TCP transfers [90, 41], and automatic buffer sizing [27, 69, 88, 39]. Changes in TCP or new congestion control schemes, possibly with cooperation from routers [53], 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 [79]. As we explain in §4.1, 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 4.1 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).

In this work, 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. 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.

Our main contribution here 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 4.3 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 [4].

Experimental results in several high BDP paths, shown in §4.4, 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 §2.2, 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 §4.5.

4.1 Socket Buffer Size and TCP Throughput

Consider a unidirectional TCP transfer from a sender \(SND\) to a receiver \(RCV\). TCP uses window based flow control, meaning that \(SND\) 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

\[
W_s = \min\{W_c, W_r, B_s\}
\]

where \(W_c\) is the sender’s congestion window \([6]\), \(W_r\) is the receive window advertised by \(RCV\), and \(B_s\) is the size of the send socket buffer at \(SND\). The receive window \(W_r\) is the amount of available receive socket buffer memory at \(RCV\), and is limited by the receive socket buffer size \(B_r\), i.e., \(W_r \leq B_r\). In the following, 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:

\[
W_s = \min\{W_c, S\}
\]

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

\[
R = \frac{W_s}{T(W_s)} = \frac{\min\{W_c, S\}}{T(W_s)}
\]

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 \(P\) 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 \(P\) is referred to as cross traffic. The forward path from \(SND\) to \(RCV\), and the reverse path from \(RCV\) to \(SND\), 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 $\rho_i$ 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 [45], we refer to the link of the forward path $P_f$ with the minimum available bandwidth $A = \min_{P_f} \{ A_i \}$ 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 (6), we can view the target transfer throughput as a function $R(S)$. Then, an important question is: given a network path $P$, 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 [79], operational handouts [94], and research papers [88], 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.

4.2 Maximum Feasible Throughput and Available Bandwidth

4.2.1 Non-congested paths

Suppose first that the network path $P$ 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[1] host at
NYU (nyu.ron.lcs.mit.edu). The capacity of the path is $C=97$Mbps, the exogenous RTT
is $T_e=40$ms, and the loss rate that we measured with ping was zero throughout our experi-
ments. We repeated a 200MB TCP transfer four times with different values of $S$. Available
bandwidth measurements with pathload [45] showed that $A$ was practically constant before
and after our transfers, with $A \approx 80$Mbps.

Figure 3 shows the throughput and RTT of the TCP connection when $S=128$KB. In this

---

1The capacity and available bandwidth measurements mentioned in this work refer to the IP layer. All
throughput measurements, on the other hand, refer to the TCP layer.
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 $R(S) = S/T_e = 25.5$ Mbps, 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., $S = A \times T_e = 400$ KB (see Figure 4). 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 ($R(S) = 79.6$ Mbps). 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 $T_a = T_e = 40$ ms.

It can be misleading to think that the previous case corresponds to the optimal socket buffer sizing, i.e., that the MFT is $\hat{R} = A$. 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 MFT will be equal to the available bandwidth. If the cross traffic

**Figure 3:** Throughput and RTT of a 200MB transfer with $S=128$ KB.
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 and make MFT larger than available bandwidth. A detailed analysis of cross traffic’s congestion responsiveness can be found in [47].

To illustrate the effect of congestion responsiveness of cross-traffic on MFT, we further increase $S$ to 550KB (see Figure 5). First point is that the transfer is still buffer limited, as it does not experience any packet losses. Second, the RTT increases by 9ms from $T_e=40$ms to $T_a=49$ms. Consequently, the throughput of the target transfer reaches $R(S) = S/T_a=89.6$Mbps, 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 6 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 obtain 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 by many as an unfair congestion behavior.

![Figure 6: Throughput and RTT of a 200MB transfer with $S=900$KB (max).](image)
4.2.2 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 be also shown analytically using a result of [77]. 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

$$R(S) \approx \min \left\{ \frac{S}{T}, f(T, p) \right\}$$

(7)

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 (7) 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 $S_\infty$ is a sufficiently large value to make the transfer congestion limited throughout its lifetime, i.e., $S_\infty > \max W_c$.

To illustrate what happens in congested paths, Figures 7 and 8 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 $C=9.7$Mbps, the RTT is $T_e=78$ms, while the available bandwidth $A$ is about 3Mbps.

In Figure 7, $S$ 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 8, on the other hand, $S$ is increased to the maximum possible value, and so the
Figure 7: Throughput and RTT of a 30MB transfer in a congested path with $S=30$KB. 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.

Figure 8: Throughput and RTT of a 30MB transfer in a congested path with $S=S_\infty$. 
4.3 **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\(^2\). Therefore, its throughput is limited by congestion window and obtains the same throughput as a regular TCP transfer. 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 [106]. 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 [16]. Second, the operating system should allow dynamic changes in the socket buffer size during the TCP transfer, increasing or decreasing it. 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

\(^2\)Maximum possible socket buffer on a host depends on OS and can modified by administrator
already in the send socket (see \[17\] §4.2.2.16). Similarly, in the case of a decrease of the receive socket buffer size, no data should be dropped. 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 \[69\] and routers in the Internet today.

4.3.1 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: \textbf{flat-rate} and \textbf{const-rate-drop}. The \textbf{flat-rate} condition is detected when the receive throughput appears to be almost constant for some time period. The \textbf{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.

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

\textbf{Figure 9:} Basic SOBAS algorithm

4.3.2 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 else the path is considered to be non-congested. Additionally, the periodic path probing allows the receiver to maintain a running average of the RTT. Each probing packet is 100B 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 §4.4, although it does make SOBAS not useful for low bandwidth paths e.g. dial-up users.

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 $\Delta=2 \times$ RTT. The measurement period $\Delta$ 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 during the RTT. If $\Delta$ 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 next section § 4.3.3, 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 $\Delta=2 \times$ 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. This 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 10). In the current implementation, throughput measurements are considered almost equal if their slope with respect to time is less than half the rate of increase in the congestion window (cwnd). For receivers using delayed ack cwnd increased rate 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% of each other). const-rate-drop happens 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. Note that const-rate-drop is expected to occur only in an underbuffered path where SOBAS will see a loss before it could saturate the path long enough to be detected as flat-rate. Since const-rate-drop occurs when a data packet is dropped after the path was saturated for two rate measurement periods, it’s very likely to be related to TCP’s congestion window. In congested paths, on the other hand, losses occur irrespective of the TCP’s congestion window and will appear in the periodic UDP packets.

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 [59] and [32]. SOBAS attempts to avoid massive slow start losses using a technique that is similar with that of [59]. 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 [25]. If later on the transfer becomes buffer limited, 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 10 shows the state diagram of the complete SOBAS algorithm. Here state 2 represent the initial slow-start phase of SOBAS. SOBAS can also get out of slow-start if it sees a rate drop without observing a flat-rate or being buffer limited. This is shown by transition 2—> 3 in the figure 10. This can happen due to losses before saturating the path. Note that this rate drop is not same as const-rate-drop as it doesn’t reveal the rate that will saturate the path. The state 6 in the state diagram represent the final state for congested path, which is inferred when SOBAS observes losses in the periodic UDP probes, and can be reached from states 2, 3 or 4. Overall, the implementation of the algorithm is roughly 1,000 lines of C code.

\[
S = R \times T
\]

4.3.3 Buffer Overflow Latency

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,

$$T(t) = T_0 + \frac{Q(t)}{C} \quad (8)$$

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

$$Q(t) = W(t) - CT_0 = C[T(t) - T_0] \quad (9)$$

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

$$\frac{dQ}{dt} = \frac{dW}{dt} \quad (10)$$

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

$$\frac{dW}{dt} = \frac{L}{T} = \frac{LC}{W} \quad (11)$$

From (10), we see that the BOL can be determined as follows

$$\int_{CT_0}^{CT_0+B} WdW = \int_0^{D_o} LCdt \quad (12)$$

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

$$D_o^{CA} = \frac{B^2 + 2BCT_0}{2LC} \quad (13)$$

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

$$\frac{dW}{dt} = \frac{W}{T} = C \quad (14)$$

Therefore,

$$\int_{CT_0}^{CT_0+B} dW = \int_0^{RSL} Cdt \quad (15)$$
which gives us the BOL in slow-start:

\[
D_o^{SS} = \frac{B}{C} \quad (16)
\]

In the presence of Delayed-ACKs, the window increase rate is reduced by a factor of two. In that case, Equations (13) and (16) should be replaced by

\[
D_o^{CA} = \frac{B^2 + 2BC T_0}{LC} \quad (17)
\]

and

\[
D_o^{SS} = \frac{2B}{C} \quad (18)
\]

respectively. Note that \( D_o^{SS} \ll D_o^{CA} \).

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 \( B/C = 500 \text{ms} \). In that case, (18) 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.

As we mentioned above, 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 10). 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

\[
4 \times T < D_o \quad (19)
\]

For \( B/C = 500 \text{ms} \), we have that \( D_o = 1 \text{sec} \) and SOBAS avoids losses as long as the RTT of the target transfer is \( T < 250 \text{ms} \). 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.
4.4 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 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.

4.4.1 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 $C=970$Mbps 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 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 [20] to emulate an RTT of 20 msec in the path.

Figures 11 and 12 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
Figure 11: With SOBAS: Gigabit path, no cross traffic ($A=950\text{Mbps}$).

Figure 12: Without SOBAS: Gigabit path, no cross traffic ($A=950\text{Mbps}$).
Figure 13: With SOBAS: Gigabit path, unresponsive traffic ($A=550$Mbps).

Figure 14: Without SOBAS: Gigabit path, unresponsive traffic ($A=550$Mbps).
Figure 15: Initial throughput, socket buffer size, and RTT with SOBAS in the path of Figure 13.

trace analysis of the two connections shows that the non-SOBAS flow experienced multiple losses at 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 afterward, about 500ms after the transfer started, SOBAS detected the \textbf{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 [75]. 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 13 and 14 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
Figure 16: With SOBAS: Gigabit path, buffer limited TCP traffic ($A=450$Mbps).

Figure 17: Without SOBAS: Gigabit path, buffer limited TCP traffic ($A=450$Mbps).
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 15 shows in more detail the initial phase of the SOBAS transfer. At the start of the transfer, SOBAS set $S=1,875$KB 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 afterward, 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 16 and 17 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.

### 4.4.2 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 than one millisecond, we again use NISTnet to create an additional delay of 10ms. Figures 18
Figure 18: With SOBAS: GaTech campus path ($A=600$Mbps).

Figure 19: without SOBAS: GaTech campus path ($A=600$Mbps).
Figure 20: With SOBAS: Path from GaTech to LBNL ($A=450$Mbps).

Figure 21: Without SOBAS: Path from GaTech to LBNL ($A=450$Mbps).
and 19 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 20 and 21 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 ($T_c = 60$ms) makes the linear window increase during congestion-avoidance in the non-SOBAS flow to be ever slower than in the previous paths.

4.4.3 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 22 and 23 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.

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 24 and 25 show the throughput and RTT for this path with and without SOBAS, respectively. The
Figure 22: With SOBAS: Path from GaTech to NYU ($A=90$Mbps).

Figure 23: Without SOBAS: Path from GaTech to NYU ($A=90$Mbps).
Figure 24: With SOBAS: Path from GaTech to NYU ($A=80$Mbps).

Figure 25: Without SOBAS: Path from GaTech to NYU ($A=80$Mbps).
Figure 26: With SOBAS: Path from GaTech to Lulea ($A=40\text{Mbps}$).

Figure 27: Without SOBAS: Path from GaTech to Lulea ($A=40\text{Mbps}$).
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 26 and 27 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.

4.5 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.
CHAPTER V

ROUTER BUFFER SIZING

The need for buffering is a fundamental “fact of life” for packet switching networks. Packet buffers in routers (or switches) absorb the transient bursts that naturally occur in such networks, reduce the frequency of packet drops and, especially with TCP traffic, they can avoid under-utilization when TCP connections back off due to packet losses. At the same time, though, buffers introduce delay and jitter, and they increase the router cost and power dissipation.

After several decades of research and operational experience with packet switching networks, it is probably surprising that we still do not know how to dimension the buffer of a router interface. As explained in more detail in §2.3, this basic question - how much buffering do we need at a given router interface? - has received hugely different answers in the last 15-20 years, such as “a few dozens of packets”, “a bandwidth-delay product”, or “a multiple of the number of large TCP flows in that link.” It cannot be that all these answers are right. It is clear that we are still missing a crucial piece of understanding, despite the apparent simplicity of the previous question.

At the same time, the issue of buffer sizing becomes increasingly more important in practice. The main reason is that IP networks are maturing from just offering reachability to providing performance-centered Service-Level Agreements (SLAs) and delay/loss assurances. Additionally, as the popularity of voice and video applications increases, the potentially negative effects of over-buffered or under-buffered routers become more significant.

Our initial objective when we started this work was to examine the conditions under which some previous buffer sizing proposals hold, to identify the pros and cons of each proposal, and to reach a compromise, so to speak. In the progress of this research, however, we found out that there is a different way to think about buffer sizing, and we were led to
new results and insight about this problem.

Specifically, there are mostly three new ideas in this work. First, instead of assuming that most of the traffic consists of “persistent” TCP flows, i.e., very long transfers that are mostly in congestion-avoidance, we work with the more realistic model of non-persistent flows that follow a heavy-tailed size distribution. The implications of this modeling deviation are major: first, non-persistent flows do not necessarily saturate their path, second, such flows can spend much of their lifetime in slow-start (even if they are “elephants” instead of “mice”), and third, the number of active flows is highly variable with time. Our results show that flows which spend most of their lifetime in slow-start require significantly less buffering than flows that live mostly in congestion-avoidance.

Second, instead of only considering link-level performance metrics, such as utilization, average delay and loss probability, we focus on the performance of individual TCP flows, and in particular, on the relation between the average throughput of a TCP flow and the buffer size in its bottleneck link. TCP accounts for more than 90% of the Internet traffic, and so a TCP-centric approach to router buffer sizing would be appropriate in practice for both users and network operators (assuming that the latter care about the satisfaction of their users/customers). On the other hand, aggregate metrics, such as link utilization or loss probability, can hide what happens at the transport or application layers. For instance, the link may have enough buffers so that it does not suffer from under-utilization, but the per-flow TCP throughput can be abysmally low.

Third, we focus on a structural characteristic of a link (or traffic multiplexer) that has been largely ignored in the past. This characteristic is the ratio of the output/input capacities. For example, a link of output capacity $C_{\text{out}}$ receives traffic from $N$ links, each of input capacity $C_{\text{in}}$, with $NC_{\text{in}} > C_{\text{out}}$. It turns out that the ratio $\Gamma = C_{\text{out}} / C_{\text{in}}$ largely determines the relation between loss probability and buffer size, and consequently, the relation between TCP throughput and buffer size. Specifically, we propose two approximations for the relation between buffer size and loss rate, which are reasonably accurate as long as the traffic is heavy-tailed. If $\Gamma < 1$, the loss rate can be approximated by a power-law of the  

\footnote{We use the terms loss probability and loss rate interchangeably.}
buffer size. The buffer requirement then can be significant, especially when we aim to maximize the throughput of TCP flows that are in congestion-avoidance (the buffer requirement for TCP flows that are in slow-start is significantly lower). On the other hand, when \( \Gamma > 1 \), the loss probability drops almost exponentially with the buffer size, and the optimal buffer size is extremely small (just a few packets in practice, and zero theoretically). Usually, \( \Gamma \) is often lower than one in the access links of server farms, where hosts with 1 or 10 Gbps interfaces feed into lower capacity edge links. On the other hand, the ratio \( \Gamma \) is typically higher than one at the periphery of access networks, as the traffic enters the high-speed core from limited capacity residential links.

We reach the previous conclusions based on a combination of experiments, simulation and analysis. Specifically, after we discuss the previous work in §2.3, we present results from testbed experiments using a Riverstone router (§5.1). These results bring up several important issues, such as the importance of provisioning the buffer size for heavy-load conditions and the existence of an optimal buffer size that depends on the flow size. The differences between large and small flows is further discussed in §5.2, where we identify two models for the throughput of TCP flows, depending on whether a flow lives mostly in slow-start (S-model) or in congestion avoidance (L-model). As a simple analytical case-study, we use the two TCP models along with the loss probability and queueing delay of a simple M/M/1/B queue to derive the optimal buffer size for this basic (but unrealistic) queue (§5.3).

For more realistic queueing models, we conduct an extensive simulation study in which we examine the average queueing delay \( d(B) \) and loss probability \( p(B) \) as a function of the buffer size \( B \), under heavy-load conditions with TCP traffic (§5.4). These results suggest two simple and parsimonious empirical models for \( p(B) \). In §5.4 we also provide an analytical basis for the previous two models. In §5.5, we use the models for \( d(B) \) and \( p(B) \) to derive the average TCP throughput and optimal buffer size, depending on the type of TCP flow (S-model versus L-model) and the value of \( \Gamma \). We also examine the sensitivity of the TCP throughput around the optimal buffer size, when buffering is necessary (i.e., \( \Gamma < 1 \)). We find out that the throughput is a robust function of the buffer size, and that even large relative
deviations from the optimal buffer size only causes minor loss in the per-flow throughput. Finally, in §5.6, we conclude by revisiting the recent debate on “large versus small buffers” based on the new insight from this work.

5.1 Experimental Study

To better understand the router buffer sizing problem in practice, we first conducted a set of experiments in a controlled testbed. The following results offer a number of interesting observations. We explain these observations through modeling and analysis in the following sections.

5.1.1 Testbed Setup

The schematic diagram of our experimental setup is shown in Figure 28. There are four

![Schematic diagram of the experimental testbed.](image)

hosts running servers/senders and four hosts running clients/receivers, all of which are Fedora Core-5 Linux systems. Each machine has two Intel Xeon CPUs running at 3.2 GHz, with 2GB memory, and with DLink Gigabit PCIexpress network interfaces. The traffic from the four senders is aggregated on two Gig-Ethernet links before entering the router. The testbed bottleneck is the Gig-Ethernet output interface that connects the router to the distribution switch. All other links in the testbed are also Gig-Ethernet and they are not utilized by more than 50%.

We use a Riverstone RS-15008 router. The switching fabric has much higher capacity than the bottleneck link, and there is no significant queueing at the input interfaces or at the fabric itself. The router has a tunable buffer size at the output line card. Specifically, we experiment with 20 buffer sizes, non-uniformly selected in the range 30KB to 38MB. With
Ethernet MTU packets (1500B), the minimum buffer size is about 20 packets while the maximum buffer size is approximately 26,564 packets. We configured the output interface to use Drop-Tail queueing, and we confirmed that the maximum queueing delay for a buffer size $B$ is equal to $B/C_{out}$, where $C_{out}$ is the capacity of the output link (1Gbps).

Two delay emulators run NISTNet [20] to introduce propagation delays in the ACKs that flow from the clients to the servers. With this configuration, the minimum RTT of the TCP connections takes one of the following values, 30ms, 50ms, 120ms or 140ms, with a different RTT for each client machine. The traffic at the output link is monitored using $tcpdump$, running on a FreeBSD 4.7 system. We record the headers of both data packets and ACKs. The packet drop rate at the monitor is 0.1%. We use these packet traces to measure link utilization and per-flow TCP throughput.

We configured the Linux end-hosts to use the TCP Reno stack with Selective Acknowledgments and with the NewReno congestion control variant. The maximum advertised TCP window size is set to 13MB, so that transfers are never limited by that window. Finally, we confirmed that the path-MTU is 1500 Bytes and that the servers send maximum-sized segments.

The traffic is generated using the open-source Harpoon system [92]. We modified Harpoon so that it generates TCP traffic in a “closed-loop” flow arrival model [87]. A recent measurement work has shown that most of the traffic (60-80%) conforms to a closed-loop flow arrival model [81]. In this model, a given number of “users” (running at the client hosts) performs successive TCP transfers from the servers. The size of TCP transfers follows a given random distribution. After each download, the user stays idle for a “thinking period” that follows another given distribution. For the transfer sizes, we use a Pareto distribution with mean 80KB and shape parameter 1.5. These values are realistic, based on comparisons with packet traces we obtained from NLANR. The think periods follow an exponential distribution with mean duration of one second. The key point, here, is that the

---

2We have also experimented with open-loop TCP flow arrivals, without observing any qualitative difference in the results.
generated traffic, which resembles the aggregation of many ON-OFF sources with heavy-tailed ON periods, is Long-Range Dependent (LRD) [102]. As will be shown next, the LRD nature of the traffic has major implications, because it causes significant deviations from the average offered load over long time periods.

One important property of the previous closed-loop flow arrival model is that it never causes overload (i.e., the offered load cannot exceed the capacity). Specifically, suppose that we have $U$ users, a flow size of $S$ bytes, a think period of $T_h$ seconds, and that the flow completion time is $T_t$. Then, the offered load generated by the $U$ clients is $\frac{US}{T_h+T_t}$. The offered load cannot exceed the capacity of the bottleneck link $C_{out}$. If that link becomes congested, the transfers take longer to complete, the term $T_t$ increases, and the offered load remains at or below $C_{out}$ [11]. Note that this is not the case in an open-loop flow arrival model, where new flows arrive based on an external random process (e.g., a Poisson process).

We control the offered load by emulating different numbers of users. The three experiments that we summarize in this work, referred to as $U_{1000}$, $U_{1200}$, and $U_{3000}$, have $U=1000$, 1200 and 3000 users, respectively. The first two experiments do not generate enough offered load to constantly saturate the output link. The third experiment, $U_{3000}$, produces an offered load that is very close to the capacity (1Gbps). The run time for each experiment is 5 minutes. To avoid transient effects, we analyze the collected traces after a warm-up period of one minute.

5.1.2 Results

5.1.2.1 Link Utilization

Figure 29 shows the average utilization $\rho$ of the bottleneck link as a function of the buffer size in each of the three experiments. First note that the utilization curves, especially in the two experiments that do not saturate the output link, are quite noisy despite the fact that they represent 4-minute averages. Such high variability in the offered load is typical of LRD traffic and it should be expected even in longer time scales. If we ignore a couple of outliers however, we observe that the experiment $U_{1000}$ can generate an average utilization
of about 60-70% (with enough buffering), $U_{1200}$ can generate a utilization of about 80-90%, while $U_{3000}$ can saturate the link.

![Figure 29: Link utilization as a function of the router buffer size for the experiments $U_{1000}$, $U_{1200}$ and $U_{3000}$.](image)

Note that there is a loss of utilization when the buffers are too small. Specifically, to achieve the maximum possible utilization we need a buffer size of at least 200KB in $U_{3000}$, and an even larger buffer in the other two experiments. The reason for the loss of utilization when there are not enough buffers has been studied in depth in previous work [9]. As we argue in the following, however, maximizing the aggregate throughput should not be the only objective of buffer sizing.

Another important observation regarding the utilization of the output link is that, even if the link is moderately loaded, there can be long time periods in which the link is practically congested. This is a direct consequence of the LRD nature of the Internet traffic [64]. For instance, consider one of the $U_{1000}$ experiments in which the 4-minute average utilization is only 68%. Figure 30 shows the fraction of time in which the link utilization is higher than 90% or 95% (i.e., heavy-load conditions) when the utilization is measured in an averaging time scale of duration $T$. For example, with $T=10$ secs, we observe that the link is practically saturated, $\rho > 0.95$, for about 7% of the time. Congestion events that last for at least 10 seconds and that occur so frequently can have detrimental effects in practice, especially

60
Figure 30: Fraction of time a link is under heavy-load (i.e., more than 90% or 95% utilized) in different averaging time scales. The average utilization over the 4-minute duration of this experiment is 68%.

if the buffer size of the link was determined based on the long-term measured utilization. This example shows that it is important that the buffer sizing process considers heavy-load conditions ($\rho \approx 1$), even when the average utilization of the link is expected to be much less than 100%.

5.1.2.2 Median Per-flow Throughput

Next, we examine the relation between per-flow throughput and router buffer size. Figures 31-34 show the median per-flow throughput for two groups of flows. One group, that we refer to as “small flows”, send about 45-50KB. The “large flows”, on the other hand, send more than 1000KB. The classification of flows as small or large is arbitrary at this point; we will return to this crucial point in §5.2.

First note that, in the case of $U_{1000}$ the median per-flow throughput generally increases with the buffer size up to a certain cutoff point. The minimum buffer size that leads to the maximum per-flow throughput can be viewed as the optimal buffer size $\hat{B}$. Note that the optimal buffer size is significantly lower for small flows compared to large flows. The experiment $U_{1200}$ gives similar results (figure 32). Second, the optimal buffer size for each flow type increases as the load increases. And third, in the saturated-link experiment
(\(U_{3000}\)), we also note that the median per-flow throughput of small flows first increases up to a maximum point that corresponds to the optimal buffer size \(\hat{B}\), and it then drops to a significantly lower value.

The above experimental results raise the following questions: What causes the difference in the optimal buffer size between small flows and large flows? Why does the per-flow throughput increase up to a certain point as we increase the buffer size? Why does it drop
after that point, at least for small flows? And more generally, what does the optimal buffer size depend on? We will answer these questions in the following sections.
5.2 Two TCP Throughput Models

The experimental results show that there are significant differences in the per-flow throughput between large and small flows. Intuitively, one would expect that this may have something to do with how TCP congestion control works. It is well known that TCP has two distinct modes of increasing the congestion window: either exponentially during slow-start, or linearly during congestion-avoidance. We also expect that most small flows complete their transfers, or send most of their packets, during slow-start, while most large flows switch to congestion-avoidance at some earlier point.

![Figure 35: Average per-flow TCP throughput as a function of flow size for a small buffer size (30KB).](image)

We first analyze the results of the $U_{3000}$ experiments to understand the relation between per-flow throughput and flow size. Figures 35, 36 and 37 show this relation for three different values of the buffer size $B$: 30KB (minimum $B$), 600KB, and 38MB (maximum $B$). Each of the points in these graphs is the average throughput of all flows in a given flow size bin. The bin width increases exponentially with the flow size (note that the x-axis is in logarithmic scale).

These graphs show that the average throughput increases with the flow size, up to a certain point. Then, at least for the two smaller buffers, the average throughput tends
Figure 36: Average per-flow TCP throughput as a function of flow size for a medium buffer size (600KB).

towards a constant value as the flow size increases (but with high variance). How can we explain and model these two distinct regimes, an increasing one followed by a constant?

One may first think that the increasing segment of these curves can be modeled based on TCP’s slow-start behavior. Specifically, consider a flow of size $S$ bytes, or $M(S)$ segments, with RTT $T$. If an ACK is generated for every new received segment (which is the case in the Linux 2.6.15 stack that we use), then the throughput of a flow that completes during slow-start is approximately 

$$R_{ss}(S) = S/[T \cdot D(S)],$$

where

$$D(S) = 1 + \lceil \log_2(M(S)/2) \rceil$$

is the number of RTTs required to transfer $M(S)$ segments during slow-start when the initial window is two segments and an additional RTT is needed for connection establishment. As we see in Figure 35, however, the slow-start model significantly overestimates the TCP throughput in the increasing phase of the curve.

A more detailed analysis of many flows in the “small size” range, revealed that a significant fraction of them are subject to one or more packet losses. Even though it is true that they usually send most of their packets during slow-start, they often also enter congestion-avoidance before completing. An exact analysis of such flows is difficult and it results in
complex expressions (see [71] for instance). For our purposes, we need a simple model that can capture the increasing segment of the average per-flow throughput with reasonable accuracy, and that can be used to derive the optimal buffer size. Therefore, we identified a simple empirical model that fits the increasing segment of the observed throughput values fairly well over a wide range of buffer sizes.

We refer to this empirical model as the $S$-model. According to the $S$-model, the average throughput of a flow with size $S$ bytes is

$$R_S(S) = \frac{S}{T[D(S) + v p M(S)]}$$

(21)

where $T$ is the flow’s RTT, $p$ is the packet loss probability, $D(S)$ is, as defined earlier, the number of RTTs required to transfer $S$ segments in slow-start, and $v$ is the number of additional RTTs that each retransmitted packet introduces. In the version of Linux that we use, which relies on SACKs, each dropped packet is usually recovered with Fast-Retransmit in a single RTT, and so we set $v = 1$. We repeat that the $S$-model is an empirical formula. Based on $ns$-$2$ simulations, we found that the $S$-model also approximates TCP Reno well.

It is based on the idea that, at least for smaller flows, most packets are transferred during
slow-start, and the transfer’s duration is increased by a number of RTTs that is proportional to the number of dropped packets.

In Figures 35-37, we plot the S-model using the average RTT and loss probability observed in each experiment. Note that the S-model is an excellent approximation to the observed average per-flow throughput up to a certain flow size, which depends on the buffer size. Actually, in the case of the maximum buffer size (Figure 37), the S-model fits very well almost all flow sizes. The reason is that, with that buffer size, the loss rate is very low and so almost all flows, including the largest ones that send more than 10,000 packets, complete during slow-start.

In the case of the two lower buffer sizes, note that the experimental average per-flow throughput curves tend towards a size-independent value as the flow size increases beyond the scope of the S-model. In that range, flows send most of their packets during congestion avoidance. There are several models for that TCP regime. We choose to use the simplest, which is the well-known “square-root model” of [70], so that the derivations of the following sections are tractable. According to that model, which we refer to as the L-model, the average throughput for a flow in congestion avoidance is:

\[ R_L = \frac{k L}{T \sqrt{P}} \]  

(22)

where \( L \) is the flow’s Maximum Segment Size (MSS). Here \( k \) is a constant that depends on the exact variant of TCP [70] (we set \( k=1.22 \)).

Figures 35 and 36 show that the L-model gives a reasonable approximation for the average throughput of large flows. The variance is high, however, and the model applies only as long as the corresponding flows send most of their packets in congestion-avoidance.

One might expect that there is a specific size threshold that separates the scope of the S-model and L-model. Note, however, that this threshold would also depend on the buffer size, because the latter controls the packet loss probability. It is the loss probability, together with the flow size, that determine whether a flow will send most its packets in slow-start or congestion-avoidance. In general, the scope of the S-model expands towards larger flow sizes as we increase the buffer size, because the loss rate decreases and more
larger flows complete during slow-start. This is an interesting observation with significant implications on how we think about TCP “mice versus elephants”. It is common that large TCP flows, say more than a few tens of KB, are viewed as “elephants” and they are modeled in congestion-avoidance. Slow-start, on the other hand, is viewed as important only for flows that send up to a few tens of packets. As the previous results show, however, the mapping of small flows to slow-start and large flows to congestion-avoidance may be misleading, especially with larger buffer sizes.

Finally, we attempted to find a quantitative criterion that can classify TCP flows as either following the S-model or the L-model. The best classifier, among many that we experimented with, is the number of congestion events that a flow experiences. A congestion event here is defined as one or more packet losses that are separated from other losses by at least two RTTs. Flows that saw at most 4 congestion events are reasonably close to the S-model, while flows that experienced 5 or more congestion events are closer to the L-model. It should be mentioned, however, that there is also a “grey region” of flow sizes that fall between the S-model and L-model and that cannot be approximated by either model. In the following, we ignore those flows and work entirely with the S-model and L-model, assuming that the former captures flows that sent most of their traffic in slow-start, while the latter captures flows that experienced at least 5 congestion events.

5.3 A Simple Case-study

In the previous section, we identified two models that express the per-flow TCP throughput as a function of the loss probability and RTT that the flow experiences in its path. In this section, we consider a TCP flow of size $S$ that goes through a single bottleneck link. The link has capacity $C$ and $B$ packet buffers. Our goal is to first derive the throughput $R(B)$ of the flow as a function of the buffer size at the bottleneck link, and then to calculate the buffer size that maximizes the throughput $R(B)$. To do so, we need to know the loss probability $p(B)$ and average queueing delay $d(B)$ as a function of $B$. As a simple case-study, even if it is not realistic, we consider the $M/M/1/B$ queueing model. Further, we

---

3The flows in the “grey region” contribute to less than 15% of bytes transferred.
focus on heavy-load conditions, when the link utilization is close to 100% for the two reasons we explained in §5.1: first, a closed-loop flow arrival model cannot generate overload, and second, the LRD nature of the traffic implies that there will be significant time periods of heavy-load even if the long-term average utilization is much less than 100%.

In the $M/M/1/B$ model, the loss probability is given by, $p(\rho, B) = \frac{(1-\rho)\rho^B}{1-\rho^{B+1}}$. In the heavy-load regime, as $\rho$ tends to 1, the loss probability becomes simply inversely proportional to the number of packet buffers $p(B) = 1/B$. The average queueing delay, in the heavy-load regime, becomes $d(B) = B/(2C)$. The RTT of the TCP flow we consider can then be written as

$$T = T_o + \frac{B}{2C} \quad (23)$$

where $T_o$ is the RTT of the flow excluding the queueing delays in the bottleneck link.

We can now substitute the previous expressions for the loss probability and RTT in the throughput equations for the S-model and L-model, (21) and (22), to derive the average throughput $R(B)$ as a function of the buffer size

$$R_S(B) = \frac{S}{(T_o + \frac{B}{2C})[D(S) + \frac{\nu M(S)}{B}]}, \quad (24)$$

$$R_L(B) = \frac{\sqrt{B}kL}{(T_o + \frac{B}{2C})}. \quad (25)$$

Figure 38 shows the throughput $R(B)$ for the S-model and the L-model, in the case of a link with $C = 1$Gbps and of a flow with $T_o = 60$ms and $S=30$pkts=45KB. Note that both TCP models have an optimal buffer size $\hat{B}$ at which the throughput is maximized.

The initial throughput increase as we increase $B$ can be attributed to the significant reduction in the loss probability. Near the optimal buffer size, the gain in throughput due to loss probability reduction is offset by an increase in the queueing delay. Beyond the optimal buffer size the effect of the increasing queueing delays dominates, and the throughput is reduced in both the L-model and S-model. Further, note the optimal buffer size is much lower in the S-model case.

It is straightforward to derive the optimal buffer size $\hat{B}_S$ and $\hat{B}_L$ for the S-model and
the L-model, respectively:

\[ B_S = \sqrt{\left( \frac{2vM(S)}{D(S)} \right) CT_o} \]  \hspace{1cm} (26)

\[ B_L = 2CT_o \]  \hspace{1cm} (27)

Interestingly, the optimal buffer size for the L-model is simply twice the bandwidth-delay product (BDP). On the other hand, the optimal buffer size for the S-model increases with the square-root of the BDP. This explains why the smaller flows that we considered in the experimental results have a lower optimal buffer size than the larger flows. For example, the optimal buffer size at a 1Gbps link with \( T_o=60 \text{ms} \) (BDP: \( CT_o=7.5 \text{MB} \)) is, first according to the S-model, 0.03\( CT_o \) (225KB) for S=10KB, 0.06\( CT_o \) (450KB) for S=100KB, and 0.15\( CT_o \) (1.125MB) for S=1MB. According to the L-model, on the other hand, the optimal buffer size is 2\( CT_o \), which is equal to 15MB!

Clearly, the optimal buffer size at a network link heavily depends on whether the link is optimized for smaller flows that typically send most of their traffic in slow-start, or for bulk transfers that mostly live in congestion avoidance. From the network operator’s perspective, it would be better if all flows followed the S-model so that routers could also have much
smaller buffering requirements.

5.4 Delay and Loss Models under Heavy-traffic

In the previous section, we derived closed-form expressions for the per-flow throughput \( R(B) \) as a function of the buffer size for the simplistic case of the \( M/M/1/B \) model. Of course in reality packets do not arrive based on a Poisson process and they do not have exponentially distributed sizes. Instead, the packet interarrival process exhibits significant correlations and burstiness even in highly multiplexed traffic [49, 64].

In this section, we aim to address the following question: In the heavy-load regime \((\rho \approx 1)\), are there simple functional forms for \( p(B) \) and \( d(B) \) that are reasonably accurate for LRD TCP traffic across a wide range of output/input capacity ratios and degrees of statistical multiplexing? Given that the exact expressions for \( p(B) \) and \( d(B) \) could depend on several parameters that describe the input traffic and multiplexer characteristics, here we focus on “functional forms”, i.e., on general expressions for these two functions, without attempting to derive the exact dependencies between the involved parameters and \( p(B) \) or \( d(B) \). For instance, a functional form for the loss probability could be of the form \( p(B) = a B^{-b} \), for some unknown parameters \( a \) and \( b \). Recall that the reason we focus on the heavy-load regime is due to the LRD nature of the traffic: even if the long-term utilization is moderate, there will be significant time periods where the utilization will be close to 100%.

The mathematical analysis of queues with finite buffers is notoriously hard, even for the simplest traffic models. For instance, there is no closed-form expression for the loss probability in the simple case of the \( M/D/1/B \) model [13]. Even asymptotic analysis (as \( B \) tends to infinity) is hard for arbitrary load conditions and general traffic models. On the other hand, it is often the case that good empirical approximations do exist in the heavy-load regime. For instance, see the Allen-Cunneen formula for the average queueing delay in the \( G/G/1 \) model [13].

The approach that we follow in this section is largely empirical and it is based, first, on extensive simulations, and second, on analytical reasoning. In particular, we examine
whether we can approximate $p(B)$ and $d(B)$ by parsimonious functional forms in heavy-load conditions. The main conclusions of the following study are summarized as follows:

1. the queueing delay $d(B)$ can be approximated as linearly increasing with $B$ (up to a certain cutoff point that depends on the maximum offered load),

2. the loss probability $p(B)$ can be approximated as decreasing exponentially with $B$ (i.e., $p(B) \approx ae^{-bB}$) or as a power-law of $B$ (i.e., $p(B) \approx aB^{-b}$), depending on the output/input capacity ratio.

Next, § 5.4.1 shows some of the simulation results that led us to these conclusions, while § 5.4.2 provides an analytical basis for these models and for the conditions under which they hold.

### 5.4.1 Simulation Results

Figure 39 shows our $ns(2)$ simulation setup. There are $N_{in}$ input links, each with capacity $C_{in}$, feeding an output link that has capacity $C_{out}$ and buffer size $B$. There are $\max(20, N_{in})$ servers that are connected to the input links with propagation delays that vary between 5ms and 45ms. The round-trip propagation delay $T_o$ in this setup varies between 30ms and 110ms, with a harmonic mean of 60ms. There are $U$ users in the system that create TCP transfers through the output link. Each user follows the closed-loop flow generation model, selecting a random server for each transfer. The transfer sizes follow a Pareto distribution with mean 80KB and shape parameter 1.5.

By choosing $N_{in}$ and $U$ as always greater than the ratio $\Gamma = C_{out}/C_{in}$, the bottleneck is always the output link. Of course, if $N_{in}C_{in} < C_{out}$ then there is no reason for buffering at
the output link. Also, $U$ is set to a point that the offered load is always enough to saturate the output link, as long as $B$ is sufficiently large. Because of the closed-loop nature of the traffic, the output link is saturated, but it is not overloaded. The simulation parameters are listed in Table 1. Note that these simulation parameters can capture a wide variety of traffic multiplexers. A residential or office access link used by a small number of people can be well represented by $N_{in} = 2$, $U = 5$ and $\Gamma = 0.1$. Similarly, the parameter setting $N_{in} = 1000$, $U = 25$ and $\Gamma = 10$ can model the upstream link of a DSLAM packet multiplexer.

**Table 1: Simulation parameters**

<table>
<thead>
<tr>
<th>$N_{in}$</th>
<th>$U$</th>
<th>$\Gamma = C_{out}/C_{in}$</th>
<th>$C_{out}$</th>
<th>$C_{in}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>0.1</td>
<td>2.5 Mbps</td>
<td>25 Mbps</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>0.1</td>
<td>2.5 Mbps</td>
<td>25 Mbps</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.1</td>
<td>50 Mbps</td>
<td>500 Mbps</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>0.1</td>
<td>50 Mbps</td>
<td>500 Mbps</td>
</tr>
<tr>
<td>1000</td>
<td>25</td>
<td>10</td>
<td>10 Mbps</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>10</td>
<td>10 Mbps</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>10</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
</tr>
<tr>
<td>20</td>
<td>500</td>
<td>10</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
</tr>
</tbody>
</table>

We show here few typical results. Figures 40 and 41 show the loss probability $p(B)$ and average queueing delay $d(B)$ for a value $\Gamma$ that is less than one. Figures 42 and 43 show the loss probability and average queueing delay for a value of $\Gamma$ that is larger than one.

**Figure 40:** Loss probability as a function of buffer size for low $\Gamma$ ($N_{in}=20$, $U=100$, $\Gamma=0.1$).
Notice how the loss probability decreases in the case $\Gamma < 1$ (Figure 40). The decrease is almost linear in a log-log plot, which means that the loss probability can be at least approximated by a power-law functional form, $p(B) = a B^{-b}$.

On the other hand, Figure 42 shows the loss probability when $\Gamma > 1$. Here, the decrease is almost linear in a linear-log plot, which means that the loss probability can be modeled by an exponential functional form, $p(B) = a e^{-bB}$. 

---

**Figure 41:** Queueing delay as a function of buffer size for low $\Gamma$ ($N_{in}=20$, $U=100$, $\Gamma=0.1$).

**Figure 42:** Loss probability as a function of buffer size for high $\Gamma$ ($N_{in}=1000$, $U=25$, $\Gamma=10$).
In terms of the average queueing delay, Figures 41 show and 43 that $d(B)$ increases almost linearly with $B$, up to a certain cutoff point. After that point, $d(B)$ becomes almost constant with $B$, meaning that the offered load that the $U$ users can generate is not enough to keep the buffer full. Increasing the buffer size beyond this cutoff point would not have a significant effect on the traffic. Consequently, we limit the scope of our loss probability and queueing delay models in the range in which the queueing delay increases almost linearly with $B$.

5.4.2 Analytical Basis for Loss Rate Models

In the following, we refer to the two functional forms for the loss rate as the EX-form $p(B) = ae^{-bB}$ and the PL-form $aB^{-b}$. The fact that the loss rate can be modeled with these two expressions should not be surprising. Previous work, for the asymptotic analysis of the tail probability with various queueing models, has shown that the tail probability can be decaying exponentially or as a power-law, depending on the characteristics of the input traffic. For instance, see [22]. We next explain how $\Gamma$ affects the tail queueing probability with simple analytical arguments for $\Gamma \ll 1$ and $\Gamma > 1$. The following should certainly not be viewed as rigorous mathematical proofs. They do provide analytical insight, however, on the EX-form and PL-form approximations for the loss rate.
Consider a FCFS output link with capacity $C_{out}$, buffer size $B$ packets, and $N$ input links with capacity $C_{in}$. To further simplify, suppose that all packets have the same size $L$. The only assumption about the input traffic is that it generates heavy-tailed burst sizes, i.e., the probability that an input link will send a burst of more than $x$ packets decays as a power-law of $x$, at least for large values of $x$. Previous measurement work has shown that TCP traffic exhibits strong burstiness and correlation structure in sub-RTT timescales [49].

$\Gamma \ll 1$: Let us assume that during a busy period of the output link, only one input link is active. Suppose that the active link sends a burst of $R$ packets. In the time that it takes to transmit a single packet at the output link, $1/\Gamma$ packets can arrive to its buffer from the active link. So, the maximum queue size at the output link will be $R(1 - \Gamma)$, which is approximately equal to $R$ because $\Gamma \ll 1$. So, because $R$ follows a heavy-tailed distribution, the queue size distribution at the output link will also follow a heavy-tailed distribution. Based on earlier results [22], we know that in that case the queueing tail probability $P[q > B]$ drops as a power-law of $B$. The loss rate $p(B)$, however, can be approximated by the queueing tail probability as long as the buffer size $B$ is not too small. So, we expect the PL-form to be a good approximation for $p(B)$ as long as $\Gamma \ll 1$, the input traffic has heavy-tailed burst sizes, and the buffer size is sufficiently large.

$\Gamma > 1$: Suppose again that an input link sends a burst of $R$ packets to the output link. The latter can transmit $\Gamma$ packets at the time it takes to receive one packet from that input, and so the queue will be always empty. So, in this case we need to consider events where several input links are active in the same busy period of the output link. Let us further assume that the $N$ input links are equally loaded and that they carry independent traffic. Say that $X$ is the number of packets that arrive at the output link during each packet transmission period $L/C_{out}$. $X$ can be viewed as a binomial random variable with parameters $N$ and $p$, where $p$ is the average utilization of each input link. For large $N$ and small $p$, $X$ can be approximated by a Poisson random variable, which means that the arriving traffic at the output link is not heavy-tailed. So, based on earlier results [22, 57], the queueing tail distribution $P[q > B]$ follows the EX-form. As previously, we can approximate the loss rate $p(B)$ by the EX-form, as long as the buffer size is not too small. In summary,
we expect the EX-form to be a good approximation for \( p(B) \) as long as \( \Gamma > 1 \), there are many, lightly loaded and independent input links, and the buffer size is sufficiently large.

The previous analytical arguments do not cover several important cases. What happens when \( \Gamma \) is less, but not much less, than one? How does the degree of “heavy-tailedness” of the input traffic affect the PL-form approximation? In the case of the EX-form, what if the number of input links is low, or if some of the input links are heavily loaded, or if there are inter-link correlations? And finally, how good are these approximations for very small buffer sizes, say less than 10-20 packets? We have examined such corner cases with a number of simulations. To summarize those results, it appears that the EX-form is quite robust as long as \( \Gamma > 1 \). On the other hand, the PL-form is not an acceptable approximation when \( \Gamma \) is less but close to one and the input traffic is not strongly heavy-tailed. In that case, neither the PL-form nor the EX-form are particularly good approximations.

### 5.5 Optimal Buffer Size

In the previous section, we proposed functional forms for the average queueing delay and loss rate. The former is a linear function of the buffer size, \( d(B) = f B/C \), up to a certain point determined by the maximum offered load. The latter is either the EX-form \( p(B) = ae^{-bB} \) or the PL-form \( aB^{-b} \). In this section, we derive expressions for (1) the average per-flow TCP throughput \( R(B) \) as a function of the buffer size in the heavy-load regime, and (2) the optimal buffer size \( \hat{B} \), i.e., the value of \( B \) that maximizes the average per-flow TCP throughput. These expressions are derived for both TCP throughput models (L-model and S-model) and for both loss rate forms (EX-form and PL-form).

#### 5.5.1 PL-form

First, we consider the case that the loss rate decreases as a power-law of the buffer size,

\[
p(B) = a B^{-b}
\]  

(28)

where \( a \) and \( b \) are positive constants. The queueing delay is modeled by a linear function, and so the RTT \( T(B) \) is given by

\[
T(B) = T_0 + f \frac{B}{C}
\]

(29)
where $T_o$ is the round-trip propagation delay (excluding queueing delays) at the bottleneck link, $C$ is the output link capacity, and $f$ is a positive constant.

### 5.5.1.1 L-model

In the L-model, the throughput $R(B)$ is given by

$$R(B) = \frac{kL}{\sqrt{aB^{-b}(T_o + f\frac{B}{C})}}$$

(30)

After setting the derivative of $R(B)$ to zero we find out that the optimal buffer size $\hat{B}$ is:

$$\hat{B} = \frac{b}{f(2 - b)}CT_o$$

(31)

The second derivative confirms that this is indeed a maximum.

Equation (31) shows that the maximum per-flow throughput is positive when $b < 2$. In our simulations, we observed that this is always the case, and that typical values for $b$ and $f$ are around 0.5 and 0.4, respectively. This makes $\hat{B}$ approximately $0.83CT_o$. Also note that the optimal buffer size is independent of the parameter $a$. What determines the value of $\hat{B}$ is the rate $b$ at which the loss rate decays with $B$, rather than the absolute value of the loss rate.

### 5.5.1.2 S-model

In the S-model, the throughput $R(B)$ is given by

$$R(B) = \frac{S}{\left[D(S) + v M(S) a B^{-b}\right] \left(T_o + f\frac{B}{C}\right)}$$

(32)

where $D(S)$, $v$, and $M(S)$ are the previously defined S-model parameters for a flow of size $S$. In the following, we set $v = 1$ (as discussed in §5.2).

Again, after calculating the first two derivatives, we find that the optimal buffer size $\hat{B}$ is the solution of the following equation:

$$\frac{ab M(S) CT_o}{\left[D(S) + v M(S) a B^{-b}\right] \left(T_o + f\frac{B}{C}\right)} = a M(S) f(1 - b) B^{-b} + fD(S)$$

(33)

Unfortunately, we do not have a closed-form solution for this equation. With the parameter values that result from our simulations, however, we observed that its numerical solution is always positive.
5.5.1.3 Remarks for the PL-form case and an example

For the $M/M/1/B$ model under heavy-load, the loss rate conforms to the PL-form with $a = 1$ and $b = 1$, and the delay coefficient is $f = 1/2$. For these parameter values, (31) reduces to $\hat{B} = 2CT_o$, while (33) gives $\hat{B} = \sqrt{\frac{2M(S)}{D(S)}} CT_o$. These are the same expressions we derived in §5.3.

Figure 44 shows $R(B)$ for the S-model and the L-model when the loss rate is modeled by the PL-form. The capacity $C$ and the propagation delay $T_o$ in this example are 50Mbps and 60ms, respectively. The model parameters for the loss rate and the queueing delay are taken from the simulation with $N_{in}=20$, $U=100$ and $\Gamma=0.1$. The flow size (for the S-model) is $S=30$ packets. Note that the optimal buffer size with the S-model is significantly lower than with the L-model (about 100 packets versus 400 packets, respectively).

![Figure 44: TCP throughput for the S-model and L-model when the loss rate is given by the PL-form.](image)

5.5.2 EX-form

In this case, the loss rate $p(B)$ is given by

$$p(B) = ae^{-bB}$$

(34)

where $a$ and $b$ are positive constants and the RTT $T(B)$ is again given by (29).
5.5.2.1 L-model

The per-flow throughput for the L-model under the EX-form is

\[ R(B) = \frac{kL}{\sqrt{ae^{-bB}(T_o + f\frac{B}{2})}}. \]  

(35)

It is easy to show that the first derivative becomes zero when

\[ B = \frac{2}{f_b}(f - \frac{bCT_o}{2}). \]  

(36)

The second derivative shows, however, that this buffer size corresponds to \textit{minimum} throughput. The buffer size that leads to maximum throughput, in this case, is either zero (given that the buffer size cannot be negative) or \( \infty \), depending on the sign of (36). Specifically, if \( dR/dB \) is negative at \( B = 0 \), then the buffer size of (36) is positive and it corresponds to minimum throughput, while the buffer size that gives maximum throughput is negative. In that case, it is best to set the buffer size to zero (\( \hat{B} = 0 \)). Otherwise, if \( dR/dB \) is positive at \( B = 0 \), the buffer size of (36) is negative, the throughput keeps increasing with the buffer size, and the optimal buffer size is, theoretically at least, \( \hat{B} \to \infty \).

With the parameter values obtained from our simulations (except when \( N_{in}=20, \, U=25 \) and \( \Gamma=10 \), the case where the offered load is too small to generate any significant queueing and loss rate), we find numerically that the optimal buffer size in this case is \( \hat{B} = 0 \).

5.5.2.2 S-model

Similarly for the S-model, the throughput is given by

\[ R(B) = \frac{S}{[D(S) + vM(S)ae^{-bB}](T_o + f\frac{B}{2})}. \]  

(37)

Setting the first derivative of \( R(B) \) to zero gives the following equation

\[ \frac{fD(S)}{vM(S)} + (af - abCT_o)e^{-bB} = abfBe^{-bB} \]  

(38)

The previous equation does not always have a unique root, making it hard to argue for the location of the global maximum of \( R(B) \). Given specific parameter values, however, it is straightforward to determine numerically the optimal buffer size \( \hat{B} \). As in the L-model case, with the parameter values obtained from our simulations (except when \( N_{in}=20, \, U=25 \) and \( \Gamma=10 \)), we find numerically that the optimal buffer size is \( \hat{B} = 0 \).
5.5.2.3 Remarks for the EX-form case and an example

Figure 45 shows $R(B)$ for the S-model and the L-model when the loss rate is modeled by the EX-form. The capacity $C$ and the propagation delay $T_o$ in this example are 100Mbps and 60ms, respectively. The model parameters for the loss rate and the queueing delay are taken from the corresponding simulation with $N_{in}=1000$, $U=500$ and $\Gamma=10$. The flow size (for the S-model) is $S=30$ packets.

![Graph showing TCP throughput for S-model and L-model](image)

**Figure 45:** TCP throughput for the S-model and L-model when the loss rate is given by EX-form.

Note that in both cases, S-model and L-model, the optimal buffer size is zero. Even though it is mathematically possible (as explained earlier) to have a non-zero, or even infinite optimal buffer size in the EX-form case, in all our simulations the optimal per-flow throughput is obtained when the buffer size is zero or very low (less than 10 packets). This is a major difference between the EX-form and the PL-form, and it reflects how important the output/input capacity ratio is in the buffer sizing problem.

5.5.3 Sensitivity Analysis

So far, we have given expressions for the optimal buffer size when the PL-form holds. The optimal buffer size with the EX-form, on the other hand, is usually zero (or close to zero),
and we will not consider it further here.

In practice, it would be difficult to fine-tune the buffer size at exactly the optimal value that (31) or (33) predict, given the estimation uncertainty in the involved parameters. Also, it may not be so important to actually achieve the optimal per-flow throughput. Consequently, an important question is: what is the relative reduction in the per-flow throughput when the buffer size is within a given fraction, say $\omega$, of the optimal size $\hat{B}$? Or, in other words, how steep is the $R(B)$ curve around the optimal point?

To answer this question, we rely on the expressions for $R(B)$ for the L-model and the S-model, in the case of the PL-form. We calculate the per-flow throughput $R(B)$ at the buffer size $B = \omega \hat{B}$. Then, we report the relative error between $R(\omega \hat{B})$ and $R(\hat{B})$:

$$e_R(\omega) = \frac{R(\hat{B}) - R(\omega \hat{B})}{R(\hat{B})}$$  \hspace{1cm} (39)

For the L-model, $e_R(\omega)$ is equal to

$$e^L_R(\omega) = 1 - \frac{2 \omega^{0.5b}}{2 + b(\omega - 1)}$$  \hspace{1cm} (40)

Note that the relative error does not depend on the delay parameter $f$. Only the loss probability decay factor $b$ matters.

Figure 46: Sensitivity of the throughput to errors in the optimal buffer size, for the L-model (top) and the S-model (bottom).
For the S-model, we do not have a closed-form expression for the optimal buffer size, and so we rely on a numerical calculation of the relative error $e_{SR}(\omega)$.

Figure 46 shows $e_{LR}(\omega)$ and $e_{SR}(\omega)$ for the L-model and S-model, respectively, as $\omega$ varies from 0.25 to 1.75. We choose three values of $b$: 0.35, 0.5 and 1.0. Recall that $b=1.0$ corresponds to the $M/M/1/B$ model. Notice that the error is higher when we underestimate the optimal buffer size rather than when we overestimate it. However, the relative error is quite low around the optimal buffer size, and it remains below 10%-15% even when we set the buffer to 40% of the optimal size.

5.6 Summary

Recently, there has been an interesting debate regarding the sizing of router buffers. In section §2.3, we summarized the key points and opinions in this debate. In this section, we put the results of this work in the context of that debate.

First, we emphasize that this work does not focus only on link utilization. Having the minimum amount of buffering to keep the utilization high is an objective that does not take into account the performance of the major transport protocol (TCP) and of most application traffic.

The work presented here provides further evidence that the buffer provisioning formula based on the BDP is probably far from optimal. In several of our simulation and modeling results, we observed that the optimal buffer size is much less than the BDP. That rule-of-thumb only applies in the very special case that the link is saturated by a single persistent TCP connection, and so it can be quite misleading in most practical cases. From this point of view, we agree with [9] that the buffer size can be significantly less than the BDP when a link carries many flows.

Previous buffer sizing research has focused on the number $N$ of large flows sharing a link [9, 23]. Practically, however, the “number of flows” $N$ is a rather ill-defined concept in the context of buffer sizing, because it is not clear which TCP flows should be included in $N$. As we show here, TCP flows can behave according to the S-model or the L-model, and that is not strictly based on their size. Even very large flows can conform to the S-model if
the loss probability is quite low.

Our results are in agreement with the earlier work by Enachescu et al. [28], which suggests that the buffer size of some links can be significantly reduced to as low as a dozen of packets. As we showed in §5.4, this is the case when the output/input capacity ratio is larger than one, and the loss probability drops exponentially with the buffer size. However, we disagree with [28] about the reasons that allow for this decreased buffer size. The buffer decrease when $\Gamma > 1$ is not related to TCP’s maximum window and it does not require TCP pacing or moderate utilization.

We observe that in some cases, especially in links where the capacity ratio $\Gamma$ is much lower than one, the buffer requirement can still be a significant fraction of the BDP, especially when the link mostly carries L-model flows. We expect these conditions to be true in some links at the periphery of the network. Special attention should be given to the edge links of server farms in the outgoing direction (e.g., from 10GigE server ports to an 1GigE edge link), and to customer access links, in the incoming direction, (e.g., from OC-48 core links to an OC-3 customer access link).

Finally, we point out that it is difficult to arrive at a simple and “handy” formula that one can use for sizing the buffers of any router interface. We hope to have conveyed that practically such a formula may not exist. The appropriate buffer size at an Internet link depends on several parameters that are related to both the offered load (flow size distribution, types of TCP traffic, etc) and to network design (capacity ratios, degree of statistical multiplexing, etc). On the more positive side, we have provided evidence that the per-flow throughput does not drop much when the buffer size deviates from its optimal value, especially when it is overestimated. A network administrator can estimate these parameters by monitoring the input capacity of arriving flows. Depending on the observed capacity ratio and the policy of favoring slow-start or congestion-avoidance flows, one can set the buffer size to a few packets ($\Gamma > 1$), a small fraction of the BDP ($\Gamma < 1$, S-model), or in the order of the BDP ($\Gamma < 1$, L-model).
Models of aggregate TCP traffic are valuable in networking research and practice. Much of the previous work in this area has been focusing on the model of persistent TCP flows, i.e., on flows that have unlimited data to send and that are not limited by the receiver advertised window. This model is mathematically tractable and it is easier to simulate, but at the same time it fails to capture key aspects of real Internet traffic [52]. Specifically, it ignores the heavy-tailed nature of the flow size distribution (that can produce Long-Range Dependency), the significant variations in the number of active flows with time, and the relation between congestion and the flow arrival process. On the other hand, some previous work has considered non-persistent TCP flows, following a heavy-tailed size distribution. The open issue there is whether the arrival process of the TCP flows should be modeled in an open-loop (OL) manner (say, according to a Poisson process), or in a closed-loop (CL) manner (say, from a number of interactive users). This work focuses on the differences that the flow arrival process, OL versus CL, causes in the generated aggregate traffic. The related issue of which model is more realistic has been the focus of a recent measurement study [81].

We start with the fluid Processor Sharing (PS) models for the OL and CL flow arrival processes. The PS models provide an accurate estimate of the offered load in light/moderate load conditions. On the other hand, when the load approaches the capacity, the PS models can lead to significant underestimation of the offered load. The main problem is that the PS models ignore packet losses and TCP retransmissions, which are a significant contribution of additional load in congested links. Nevertheless, the PS models show clearly that the open-loop model can be unstable, while the closed-loop model is always stable, as the number of active flows is bounded.

We then compare the queueing performance of the (packet level) OL and CL models,
examining the loss rate and queueing delay distribution that the two models produce under the same offered load. The OL model produces higher loss rate and queueing delays than the CL model. To explain this difference, we examine the traffic variability produced by the two models in a range of timescales (10msec-1sec). We find out that the OL model results in higher variance than the CL model, especially when the timescale exceeds the TCP Round-Trip Time (RTT). The cause of the increased traffic variability in the OL model is that the latter does not reduce the flow arrival rate upon congestion. This leads to more significant overload events, in magnitude and duration, than the CL model, generating higher traffic variability. The CL model responds to congestion roughly one RTT after its occurrence, which explains why the variability difference becomes significant when the timescale is larger than the RTT.

We also examine the distribution of the number of active flows with each flow arrival model. Here, we find that the OL model results in higher variability in the number of flows than the CL model when the offered load is significant. There are time periods in which the number of ongoing flows with the OL model is much higher than the average. This observation is related to an earlier study by Schroeder et al. which showed that job scheduling is crucial mostly with the OL model, as the former gives a wider leeway to the scheduler than the CL model [87].

Finally, we focus on the transient response of the two models in terms of the congestion responsiveness of the aggregate traffic. With OL flow arrivals, the resulting traffic is not congestion responsive, meaning that the offered load does not follow the available capacity in the network. With CL flow arrivals, on the other hand, the traffic is congestion responsive.

The rest of the chapter is organized as follows. Section 6.1 reviews previous work and the limitations of the persistent flows model, respectively. Section 6.2 describe the OL and CL models and review basic results about the corresponding PS models. Our simulation setup is presented in Section 6.3, while the queueing and offered load differences between the two models are presented in Sections 6.4 and 6.5. Section 6.6 examines the variability in the offered load at different timescales, while Section 6.7 shows the variation in the number of active flows. Section 6.8 focuses on the congestion responsiveness of the two models. We
conclude in section 6.9, also discussing some implications of this work in various areas of networking research and practice.

6.1 Critique of the Persistent Flows Model

It is common for analytical and simulation studies to model most of the traffic with persistent TCP connections. A common argument to justify this model is that, typically, most traffic in an Internet link is carried by a few large TCP flows (“elephants”) and so those flows can be modeled as persistent. The smaller flows, referred to as “mice”, do not contribute a significant amount of traffic and so they are often ignored, or they are viewed as a source of stochastic noise in simulation studies. The previous argument is an oversimplification and it ignores two key characteristics of real Internet traffic. First, the size of TCP flows follows a continuous and heavy-tailed distribution in practice, rather than a bimodal distribution in which flows are either very short (mice) or very long (elephants). In other words, the previous argument ignores the flows of significant, but not extreme, size. Second, flows with very large size (relative to other flows in the aggregate) do not always have very long duration. Some large flows get higher throughput, and so their duration can be comparable to that of short flows. Such flows cannot be modeled as persistent, especially when the timescale of interest (for example, the duration of the simulation study) is longer than their duration.

To illustrate these issues, we analyze a packet trace that was collected at the border router of Georgia Tech in January 2005. The trace duration is two hours and the monitored link carries the inbound traffic in a Gigabit Ethernet segment that connects the campus network to the SoX GigaPoP. The objective of this traffic analysis is to examine the assumptions behind the persistent flows model. Note that similar studies have been conducted several times in the past (for instance, see [29]), using traces from many links and under diverse load conditions.

We first looked at the flow size distribution. We find that the C-CDF of that distribution shows clear linear decrease in a log-log plot, pointing to the heavy-tailed Pareto distribution (with shape parameter about 1.3). We also examined the distribution of flow interarrivals.
Figure 47: The fraction of bytes $f$ generated by flows that are active for the entire duration of a given time interval $T$ as a function of $T$. The error bars depict the minimum and the maximum values of the fraction $f$.

When the interarrivals are larger than 100msec or so, they can be modeled as exponential and independent (pointing to a Poisson flow arrival process). However, there are significant correlations in lower interarrivals, probably due to the generation of simultaneous flows by the same application session.

Next, we measured the number of active flows as a function of time for different flow size thresholds. If we only consider flows that are larger than 1.5MB (or $\sim$1000pkts), the number of active flows remains almost constant with time. This observation, however, should not be interpreted as validation of the persistent flows model. The reason is that even though the number of (sufficiently large) active flows remains roughly constant with time, \textit{the set of active flows changes significantly with time}. To illustrate this point we examined the fraction of bytes $f$ that is generated by flows that remained active throughout a given time interval of length $T$. With the persistent model, all flows are active throughout $T$ and this fraction should be close to 100%. If all flows lasted for less than $T$ seconds, then this fraction should be zero. We measured $f$ for the following values of $T$: 7.5, 15, 30, 60 and 120 minutes. For each value of $T$ (except for 120 minutes), we obtained 30 samples of the fraction $f$, ignoring the first two minutes of the trace, and considering flows that last longer
than 0.95T as active throughout the duration $T$. Figure 47 shows the mean, the minimum and the maximum value of $f$ as a function of $T$. The key observation here is that even for time intervals that last only 5-10mins, the fraction of traffic from persistent flows is only 40-70%. So, the assumption that the same set of flows carries almost all traffic ignores the variability due to the dynamic flow arrival and completion processes.

### 6.2 Two Models of Non-Persistent Flow Arrivals

In this section, we describe two basic models of non-persistent flow arrivals: the OL and CL models. Both models are simple and well-studied in the performance evaluation literature. This section is mostly a review of known results.

Note that the terms “open-loop” and “closed-loop” have been previously used to distinguish between non-TCP traffic (viewed as open-loop because packets arrive randomly based on an exogenous process) and TCP traffic (viewed as closed-loop because the flow is regulated by TCP congestion control). In this work, both OL and CL models describe an aggregate of TCP flows. They differ, however, in the higher level process, operating at the session or application layer, that generates these flows. Figure 48 shows a schematic diagram of the flow generation process. If the session layer uses some negative feedback from the network, so that it slows down the generation of new flows upon congestion, the resulting traffic will be closer to the CL model. Otherwise, in the absence of such feedback, the OL model is more appropriate.

#### 6.2.1 Open-Loop model

In the OL model, users or applications generate flows independent of any previous flows they may have generated. To motivate this model, consider the access link of a Web server. In the outbound direction, the server sends files to a large population of users located anywhere in the Internet. Assume that a user does not return to this server, at least for a long time, after completing a file transfer. Consequently, the server’s sessions are always with new users. If the link becomes congested, the arrival rate of new sessions will not be affected, as Internet users are typically unaware of the network state in a given path.

Consider a PS server with capacity $C$ (bytes/sec), average flow arrival rate $\lambda$ (flows/sec),
Figure 48: The flow arrival process is controlled by the session/application layer. Is that layer responsive to network congestion?

and average flow size $S$ (bytes). We refer to this model as PS-OL. The average offered load in the server is given by $\lambda S$ and the normalized offered load is

$$\rho_o = \frac{\lambda S}{C}. \quad (41)$$

If $\rho_o < 1$, the server is stable and $\rho_o$ is the average utilization. For a Poisson flow arrival process, it can be shown that the average number of active flows is given by $[58]$

$$\bar{N}_o = \frac{1}{(1 - \rho_o)}. \quad (42)$$

Otherwise, if $\rho_o > 1$, the server is unstable (as long as flows are never aborted). Since both the flow arrival rate and the average flow size are independent of the network state, the average offered load remains constant even in the presence of congestion. Further, the expected throughput of a new flow in the PS-OL model is given by the available capacity in the server,

$$\bar{R} = C(1 - \rho_o). \quad (43)$$

Deviating from the PS model, we can consider a packet-level model of a First-Come-First-Served (FCFS) queue with a finite buffer and with flows that are controlled by TCP congestion control (TCP-OL). Notice two important differences between the TCP-OL model and the PS-OL model. In the former, we can have packet drops. TCP reacts to them with retransmissions, which effectively increase the size of the affected flows. Further, it is well
known that TCP can generate redundant retransmission. This means that the actual offered load by a set of TCP flows in the OL model can be higher than what the PS model predicts in Equation (41). Second, a TCP flow can be active even when it does not compete for available capacity, because of window limitations due to slow-start, retransmission timeouts, limited advertised window, etc. This means that the average number of active TCP flows can be much larger than Equation (42).

6.2.2 Closed-Loop model

To illustrate the CL model, consider the access link of a small enterprise with, say $N$, users. In the inbound direction, most of the traffic at the link is downloads that are generated by the activity of these $N$ users. In the simplest model, each user can be in the “Active” state downloading a file, then spending some time in the “Idle” (or “Thinking”) state, and then either downloading another file, or leaving the system for a longer time period (“Inactive” state). This link would not carry more than $N$ active flows at any time. Furthermore, if the link becomes congested, then the download latencies of all active flows will increase, reducing the rate with which new flows are generated.

In the PS version of the CL model, we have a fixed number of users $N$. Each user goes through cycles of activity, with flows of average size $S$, followed by idle periods of average length $T_i$. The average session arrival rate in the CL model is given by

$$\lambda_c = \frac{N}{T_t + T_i}$$

where $T_t$ is the average flow transfer latency. The latter depends on the load at the PS server. Thus the average server utilization at the PS-CL model is given by

$$\rho_c = \frac{NS}{C(T_i + T_t)}.$$  

The average number of active flows in the PS-CL model is given by (see [11])

$$\bar{N}_c = \frac{a}{1 - a} \quad \text{for} \quad a \ll 1$$

$$= N \left(1 - a^{-1}\right) = N - \frac{CT_i}{S} \quad \text{for} \quad a > 1$$

$$\text{(46)}$$
where the normalized offered load is given by

\[ a = \frac{NS}{CT_i}. \]  

(47)

Note that the expected number of active flows for \( a \ll 1 \) is same with the OL model. On the other hand, when \( a > 1 \), \( \bar{N}_c \) increases slowly with \( a \) and remains bounded by \( N \).

Similar to the TCP-OL model, the CL model with a FCFS queue and TCP flows (TCP-CL) can deviate significantly from its PS-CL counterpart. First, as in TCP-OL, we need to consider the extra load due to required or redundant retransmissions. Second, as in the TCP-OL model, TCP is not able to always use the available capacity.

### 6.3 Simulation Setup

The previous section reviewed well-known results for the OL and CL models, based on the PS model. In this work, we are more interested in TCP-specific effects that cannot be captured by the PS model, as well as on the variance of the resulting aggregate traffic. For these reasons, we rely mostly on simulation.

Figure 49 shows our NS-2 simulation setup. There are 10 input links, each with capacity 1Gbps, connected to an output link with capacity \( C = 50Mbps \) and buffer size \( B \). This topology describes a scenario in which the bottleneck is the ingress link of an enterprise network, and where the server, backbone and client links are over-provisioned. In this setup, we have 20 servers that are connected to the bottleneck with 1Gbps links and with propagation delays that vary between 5msec and 45msec. The round-trip propagation delay in this setup varies from 30msec to 110msec, with a harmonic mean of about \( T_0 = 60msec \). We use harmonic mean as representative RTT as recommended by Dhamdhere and Dovrolis [23].

![Simulation setup](image-url)
In all simulations we use the SACK-enabled NS-2 TCP module *sack1*. The buffer size $B$ is set to the bandwidth-delay product of the path (250 packets), considering $T_0$ as the representative delay. The maximum advertised window is set to 256 packets. In the CL simulations, there are $N$ clients that initiate TCP transfers. These users arrive for the first time at the network at a random instant during the first few seconds of the simulation. After arriving, each user follows the CL flow generation process selecting a server for each transfer randomly from the set of 20 servers. In the OL simulations, the flow arrival process is Poisson with arrival rate $\lambda$. In all simulations, the flow size follows a Pareto distribution with a mean of 25 packets and shape parameter 1.5. The think time $T_t$ for the CL model follows an exponential distribution with a mean of 2 seconds. The values of $\lambda$ and $N$ are varied to obtain different offered loads in the OL and CL models, respectively. Each simulation runs for 1000 seconds and we report results for the period from 200 to 950 seconds.

### 6.4 Controlling the Offered Load

To compare the traffic characteristics and queueing performance of the OL and CL models, we first need to make sure that their parameters are selected so that both models produce *equal average offered load*. The offered load is defined as the amount of traffic that *arrives* at the bottleneck link per unit of time, and it includes traffic that may get dropped due to congestion.

Controlling the offered load in the OL and CL models, however, is not trivial. Suppose that we want to generate a certain offered load $X$ at the bottleneck link of the previous simulation setup. Given the average flow size $S$ (and the average think time $T_t$ in the case of the CL model), a common approach is to rely on the PS model. For the OL model, we can calculate the required flow arrival rate as $\lambda = X/S$. For the CL model, however, the term $T_t$ depends on the given load conditions. A crude approximation is to assume light load conditions ($a \ll 1$), and thus $T_t \ll T_i$. Then, the required number of users is $N = X T_t / S$.

Next, we examine the relation between the average offered load $X$ predicted by the two PS models, as previously described, and the actual offered load that we observe in
Figure 50: The offered load with the TCP-OL and TCP-CL models (simulated, y-axis) as a function of the offered load $X$ that is predicted by the corresponding PS models (calculated, x-axis).

Simulations with TCP traffic (TCP-OL and TCP-CL models). Figure 50 shows the results of this comparison. The capacity lines $X = C$ and $Y = C$ are shown for reference. We observe that the offered load with the TCP-OL model is very close to the load $X$ predicted by the PS-OL model, as long as $X$ remains below the capacity $C$. As $X$ approaches $C$ the TCP-OL offered load starts deviating from $X$, and when $X > C$ (overload) the TCP-OL offered load is significantly higher than $X$. The reason is that the TCP-OL offered load includes retransmissions (required or redundant) of dropped packets. The fact that the increase rate of the TCP-OL offered load drops as $X$ goes more deeply into overload is due to the increasing frequency of retransmission timeouts that the TCP connections experience. Nevertheless, the important observation here is that we can use the offered load predicted by PS-OL model as a reasonable approximation, as long as $X < C$.

In the case of the CL model, the offered load predicted by the PS-CL model is lower than that with TCP-CL, even in light/moderate load conditions. The reason, of course, is that we have ignored the load-dependent transfer time $T_t$, assuming that it is much less than $T_i$. Especially for TCP flows, however, we cannot ignore that for a flow of size $S$ there is a minimum transfer latency of several RTTs due slow-start, even if there are no queueing
delays or packet losses. Thus, we next consider the following improved approximation of the offered load with the PS-CL model,

\[ X = \frac{NS}{Ti + T_{t,min}(S)} \]  

(48)

where \( T_{t,min}(S) \) is the minimum latency required by TCP to transfer a flow of size \( S \) using slow-start. It is simple to estimate this parameter as long as the RTT and the TCP variant used are known. We refer to this approximation as the PS-CL model with a constant term for the minimum transfer time of the average flow size, or PS-CL-T for short. Figure 50 shows the relation between the offered load of the TCP-CL and PS-CL-T models (with \( T_{t,min}(S)=0.36 \text{sec} \) in our simulations). Note that the latter is a reasonably good approximation both when the link is not congested \((X < C)\) and in overload \((X > C)\). The reason the offered load is slightly above the capacity in overload is again the presence of some TCP retransmissions. In summary, the PS model can provide a reasonable approximation of the offered load in the TCP-CL model, as long as we consider the minimum transfer time with TCP slow-start for the average flow size.

**Figure 51:** The offered load from the CL model tends to that of the OL model as \( N \) and \( Ti \) increase.

Notice that the OL model can be viewed as the asymptotic limit of the CL model, if we let the number of users \( N \) and the average idle time \( Ti \) go to infinity, while the initial
transfer of each user is randomly placed on the time axis. Indeed, we may wonder whether
the offered load with the TCP-CL model approaches that of the TCP-OL model as we
increase \( N \) and \( T_i \). Figure 51 shows the offered load from the TCP-CL model for two values
of \( T_i \), 2 and 20 seconds. Note that an increase in the idle time also requires an increase in the
number of users in order to attain the same offered load. For example, with \( T_i = 2 \) sec we need
400 users to get 46Mbps of offered load, while with \( T_i = 20 \) sec we need 3200 users. We see
that the offered load between the three curves differs mostly in overload, as expected. As we
increase \( T_i \) and \( N \), the TCP-CL curve approaches the TCP-OL curve, implying the gradual
convergence of the CL model to the OL model. Notice however that this convergence is very
slow and in practice we would need a very large number of users before we can claim that
the a closed population of users can be modeled with the OL model, in overload conditions.

In the following, we use the offered load that is calculated from *ns-2* simulations.

### 6.5 Queueing Performance

Next, we compare the queueing performance of the TCP-OL and TCP-CL models. The
main observation is that, under the same offered load, the TCP-OL model results in higher
queueing delays than the TCP-CL model. If there are packet losses, then the loss rate with
TCP-OL is also higher than with TCP-CL.

Figure 52 and 53 show the loss rate and the queueing delays for the TCP-OL and TCP-
CL models as a function of the offered load. For queueing delays, we report the median
and the 90-th percentile of the per-packet delay distribution. The differences are of course
minor for light load conditions, when the offered load is, say, below 50% of the capacity.
In heavier load conditions, however, the differences are significant and cannot be ignored.
In the next section we explain these differences examining the statistical variability of the
aggregate traffic in different timescales.

### 6.6 Traffic Variability at Different Timescales

The results of the previous section suggest that the TCP-OL model produces larger traffic
burstiness than the TCP-CL model. In this section we aim to further understand what
causes this difference and to identify the load conditions and timescales in which this is
Figure 52: The loss rate as a function of the offered load for the TCP-OL and TCP-CL models.

Figure 53: The median and 90-th percentile of the queueing delay distribution as a function of the offered load for the TCP-OL and TCP-CL models.

more evident.

Figure 54 shows the variance of the offered load for an averaging timescale of 10msec, 100msec and 1sec. First, notice how the variance depends on the offered load. The variance increases up to a certain point (20-45Mbps, depending on the timescale and the model). After that point the variance decreases with the offered load. For an explanation of this
well-understood trend we refer the reader to [46, 93]. What is more relevant here is that the TCP-OL model produces higher variance than the TCP-CL model in moderate/heavy load conditions. Since the round-trip propagation delays in our simulation topology vary from 30msec to 110msec, we view the timescale of 10msec as below the typical RTT, 100msec as roughly equal to the RTT, and 1sec as larger than the RTT. The results of Figure 54 also suggest that the difference in the variance of the two models is more significant when the timescale is around the RTT or higher.

In light load conditions the two models are practically equivalent, as there is no significant queueing or packet losses and transfers are only limited by TCP’s slow-start. As the offered load increases beyond roughly 50% of the capacity, congestion episodes start to occur. In the TCP-OL model, new flows arrive independent of whether the bottleneck is congested or not. In the TCP-CL model, when a flow slows down because of congestion it also delays the generation of the next flow from the same user. This reduces the duration and magnitude of congestion events, leading to lower traffic variability than in the TCP-OL model. The response latency of the TCP-CL model cannot be faster than TCP’s RTT however; this explains why the two models “look” the same in sub-RTT timescales.

![Graph](image)

**Figure 54:** Variance of the offered load with the TCP-OL and TCP-CL models for three averaging timescales.

To further illustrate the previous explanation, Figure 55 shows the fraction of time
the offered load exceeds the link capacity in four averaging timescales. Here we see that in the sub-RTT timescale of 10msec, both models experience overload for practically the same fraction of time. When we examine the traffic at higher timescales than the RTT, however, we confirm that the TCP-OL is overloaded more frequently. The TCP-CL model experiences overload less often because its flow arrival rate reduces upon the occurrence of packet losses. Since the two models have the same average offered load, the higher overload frequency in TCP-OL is compensated with time periods in which the TCP-OL offered load is less than that in TCP-CL. These wider fluctuations make the variance of TCP-OL higher, as long as the the offered load and timescale are sufficiently large.

It is not just the frequency of overload events that differs between the two models, but also their duration. This is shown in Figure 56, where we plot the CDF of the duration of overload events at different timescales for an average offered load of 47.5Mbps. This duration is measured as the number of consecutive time periods (with length equal to the averaging timescale) in which the offered load is higher than the capacity. In the sub-RTT timescale both models have the same distribution. As the timescale increases, however, the gap between the two distributions increases, as the TCP-OL model is unable to self-regulate its offered load below the capacity. For instance, when we look at the traffic in successive

Figure 55: Fraction of time the offered load is greater than the capacity for four averaging timescales.
6.7 Number of Active Flows

In this section, we examine the number of active flows created by the TCP-OL and TCP-CL models. We show that the number of active flows in these two TCP models is much larger than that predicted by the PS model, and that TCP-OL produces higher variability in the number of active flows than the TCP-CL model, in heavy load conditions. The latter implies that the per-flow throughput in the TCP-OL model is also less predictable.

Figure 57 shows the CDF of the average number of active flows when the offered load is 70% and 95% of the capacity. The number of active flows is averaged over 1-sec intervals. We first note that the number of active flows in both models is much higher than that predicted by the processor sharing model (see Equations 42 and 46). Specifically, the PS-OL model predicts about 3 and 20 flows for offered load 70% and 95%, respectively. The corresponding averages from the TCP-OL simulations are 70 and 131. For the PS-CL model, on the other hand, Equation 46 predicts an average of 102 active flows for 95% offered load, while the average from the TCP-CL simulations is 115. These differences can be attributed to the
Figure 57: The CDF of the average number of active flows, measured at 1-sec intervals, from the TCP-OL and TCP-CL models when the offered load is 70% and 95% of the capacity.

fact that, with TCP, there is a large number of small flows that are not always competing for available capacity because of slow-start, retransmission timeouts, or other limitations.

Also notice that the TCP-OL model results in much higher variability in the number of active flows in heavy load conditions. Again, this is because the TCP-OL model does not reduce the flow arrival rate upon congestion. The number of active flows in the TCP-CL model, on the other hand, is always bounded by $N$. The increased variability in the number of active flows with the TCP-OL model means that the per-flow throughput with that model is less predictable than with TCP-CL.

6.8 Congestion Responsiveness

So far we have focused on the steady-state behavior of the two models. In this section, we examine their transient response to individual congestion events.

We refer to a traffic aggregate as congestion responsive if it reduces its offered load upon overload to a point that there is no longer congestion. The specific congestion event that we consider here is a periodic UDP stream with rate that is higher than the available capacity in the bottleneck. Given that the UDP stream does not react to congestion, the event that we simulate represents a sudden reduction of the available capacity for the TCP aggregate.
from $C$ to $C' = (1-f)C$, where $fC$ is the rate of the UDP stream. In the following, we make the offered load $\rho C$ before the congestion event to be at the same level in the TCP-OL and TCP-CL models. We set $1 - \rho < f < 1$, so that the bottleneck becomes congested when the UDP stream starts.

![Figure 58](image_url)

**Figure 58:** The response of the traffic aggregate in the TCP-OL and TCP-CL models, when a congestion event is caused by a UDP stream of rate 15Mbps. The capacity is 50Mbps and the offered load (before the congestion event) is 47.5Mbps.

Figure 58 shows the offered load from the two traffic models in 1-sec intervals. The congestion event is caused by a 15Mbps CBR UDP stream and it lasts from 200sec to 275sec. The effects of the congestion event can be examined in three stages: first, just after the congestion event starts, second, during the congestion event, and third, after the congestion event finishes.

Before the start of the congestion event, both TCP-OL and TCP-CL have the same average offered load. Their response when the UDP stream starts is that, because of TCP’s congestion control, the traffic from both models drops at a level that is close to the new available capacity (35Mbps). The similarity between the two models, however, ends there. A few seconds later the offered load in the TCP-OL model starts increasing, as more and more new flows arrive and compete for throughput. The offered load in the TCP-CL model, on the other hand, is self-regulated at the level of the available capacity, because a new flow
cannot start unless an existing flow has completed. Thus, the number of active flows in the TCP-OL model keeps increasing, while the corresponding number in the TCP-CL model stays roughly the same (also see Figure 59).

Finally, after the congestion event ends, the offered load from both models increases to capture the available capacity that has been released by the UDP stream. In the TCP-CL model, this process is completed within a few seconds. In the TCP-OL model, however, there is a large backlog of active flows that needs to be cleared before the offered load returns at its pre-congestion level. As Figure 59 shows, this effect lasts for hundreds of seconds (this depends of course on the duration and magnitude of the congestion event and on the TCP offered load before congestion). Figure 59 further shows the queueing delay in the bottleneck with each model. Notice that even though the congestion event ends at $t=275\text{sec}$, the queue remains almost full for hundreds of seconds with the TCP-OL model.

![Figure 59: The time series of the number of active flows and of the queueing delay with the TCP-OL and TCP-CL models when a congestion event is caused by a CBR UDP stream.](image)

Even if the long-term offered load at a link is below the capacity, there can be overload events that last for a few tens of seconds. The important lesson from the previous discussion is that during such events an open-loop traffic aggregate is effectively congestion unresponsive despite the fact that it consists of TCP flows. Further, the consequences of an externally imposed congestion event (say a large UDP stream or a DOS attack) can last for much
longer than the duration of the event itself, if the traffic is open-loop.

6.9 Discussion

In this work, we examined two basic models of non-persistent TCP flow arrivals, and explained how they lead to different traffic characteristics, in terms of offered load, variability in different timescales, queueing performance, number of active flows, and congestion responsiveness. In the following, we discuss some more implications of this work in other areas of networking research and practice.

**AQM and network stability:** Active queue management (AQM) mechanisms, such as RED, REM, PI controllers, etc., have been proposed as a way to stabilize congestion control. It is important to note that such studies assume persistent TCP connections. With that model, AQM mechanisms can control the queue length and the bottleneck link utilization. The effectiveness of AQM mechanisms with non-persistent traffic, however, is much less understood. The offered load of TCP-OL traffic does not depend on network state. AQM mechanisms cannot regulate such an aggregate, and they are unable to avoid persistent overload if the offered load exceeds the capacity.

**Is admission control necessary?** Several researchers advocate the use of admission control as the only way to regulate the offered load and avoid congestion collapse. We agree, if the traffic is mostly OL. Without admission control, the only way to avoid congestion collapse is to expect that users will be impatient and abandon slow ongoing transfers. Admission control can limit the number of active sessions or flows in the network. Admission control may not be necessary, however, if most of the traffic follows the CL model.

**TCP-friendly congestion control:** The use of TCP-friendly congestion control has been encouraged in all non-TCP protocols and applications. The basic motivation for such proposals is that TCP-friendly transfers can avoid congestion collapse. It should be clear however, that even if a traffic aggregate consists entirely of TCP flows, it can still cause congestion collapse or persistent overload if it is OL. The same is obviously true for TCP-friendly traffic. Therefore, the use of TCP-friendly congestion control is not sufficient to guarantee stability.
**Traffic engineering and network provisioning:** Traffic engineering, as well as other provisioning mechanisms, require an estimate for the offered load between any ingress/egress pair. Furthermore, such mechanisms assume that if a given traffic aggregate is switched from one route to another, then the throughput of that aggregate will not change. This assumption is not true for TCP-CL traffic. The offered load from such aggregates depends on the RTT and loss rate in the underlying path. On the other hand, the offered load from TCP-OL traffic does not depend on the underlying path (ignoring retransmissions), making such traffic consistent with common assumptions in traffic engineering.

**Session layer congestion control:** At the more practical side, we recommend that all network applications use some form of congestion control at the session layer. This can be as simple as adopting one of the following rules: do not generate a new session until the previous session has completed, slow down the generation of new sessions if the network is congested, or do not keep more than a certain number of sessions active. It is also important that session layer congestion control is implemented in applications that generate transfers automatically, without user intervention. For example, NNTP servers transfer news to their peers periodically, independent of whether the underlying network is congested or not. Effectively, such applications generate TCP-OL traffic.

**Incentive for congestion responsive applications:** We note that making applications congestion responsive, may limit their share of network resources. However, such a mechanism is required to keep the network stable. We argue that the same incentives that lead users and applications to adopt TCP instead of UDP (despite that this may not be the optimal protocol from the selfish perspective), will also act to promote the use of closed-loop behavior at the session layer.
CHAPTER VII

MEASURING CONGESTION RESPONSIVENESS

The stable and efficient operation of the Internet is based on the premise that traffic sources reduce their rate upon congestion. If a link has capacity $C$, then the offered load at that link should not exceed $C$ for any significant period of time (relative to the maximum possible buffering in the bottleneck link). Individual TCP flows are congestion responsive thanks to the well-known TCP congestion control/avoidance algorithms. Can we expect the same, however, for streams of non-persistent TCP flows? This depends on the characteristics of the random process that generates new flows (or sessions of flows). In this work, we rely on two well-known models to describe the flow/session arrival process at a link $L$: the “open-loop” and “closed-loop” models. The former does not generate congestion responsive traffic, while the latter does. We use these models to passively estimate the congestion responsiveness of the aggregate traffic at an Internet link.

In the flow-based open-loop model, new flows arrive independent of the load at $L$, for example according to a Poisson process. The average offered load in the open-loop model is given by $\lambda S$, where $\lambda$ is the average flow arrival rate and $S$ is the average flow size. The normalized offered load is $\rho_o = \lambda S/C$. If $\rho_o < 1$ the link is stable and $\rho_o$ is its average utilization. Otherwise, if $\rho_o \geq 1$, the link becomes unstable since the number of active flows can grow without bound. If users are impatient and abort ongoing flows after a certain time period, then the number of active flows in the underlying system will remain finite, thereby making the system stable. However, aborted flows result in user dissatisfaction and poor performance [104]. Therefore, an open-loop traffic aggregate can cause instability and/or aborted flows, even if all flows use TCP. In other words, TCP congestion control cannot avoid persistent overload when flows arrive according to the open-loop model.

In the flow-based closed-loop model, we have a fixed number of users $N$. Each user goes through a cycle of activity, with a flow of average size $S$ followed by an idle period of
average length $T_i$. The average flow arrival rate is given by $\lambda_c = N/(T_i + T_i)$, where $T_i$ is
the average flow completion time. The latter depends on the load at the link, as well as
on TCP (e.g., on the slow start algorithm). When $\rho_c =\lambda_c S/C \ll 1$, users spend most time
thinking (i.e., $T_i \ll T_i$), and the system behaves similar to the open-loop model with arrival
rate $\lambda_o = N/T_i$. However, when $\rho_c$ approaches or exceeds 1, the number of active flows in the
server increases, reducing the average per-flow throughput and increasing $T_i$. The increase
in $T_i$ reduces the flow arrival rate, keeping the offered load close to the capacity, i.e., $\lambda_c S \approx C$. 
This means that the closed-loop traffic model is always stable and it cannot cause overload.

A direct way to measure the congestion responsiveness of the traffic at an Internet link
would be to cause a short-term congestion event, and then examine whether the flow arrival
process is affected. Creating congestion events to measure the responsiveness of the traffic,
however, is a highly intrusive experiment and it is often not even possible. In Section 7.2, we
describe an indirect procedure to estimate the congestion responsiveness of the aggregate
traffic at a link by classifying each session as either open-loop or closed-loop, and then
measuring the fraction of traffic that follows the latter. We refer to this fraction as the
Closed-loop Traffic Ratio (CTR). A higher CTR value implies more congestion responsive
traffic.

Our CTR estimation technique is based on the analysis of the interarrivals of packets,
flows, and sessions. At this point, the technique is mostly applicable to well understood
client-server applications, such as HTTP/HTTPS, FTP, news and email. An extension to
peer-to-peer applications is work-in-progress. Therefore, our CTR estimates at this point
cover only 30%-80% of the traffic, depending on the measured link. Measurements at a
dozen of Internet links show that the CTR is usually between 60-80%. Such high CTR
values suggest that a strong reason behind the congestion responsiveness of Internet traffic
is that users and applications respond to congestion by slowing down the generation of new
flows/sessions. TCP’s congestion control and capacity over-provisioning are not the only
reasons we do not see significant congestion events in most of the Internet today.
7.1 Congestion Responsiveness

Traditionally, congestion control has been viewed as a function of the network or transport layer at the OSI stack. The TCP feedback loop regulates the offered load (send-window) of a connection, based on the presence of congestion in the network (see Figure 60). The previous view, however, ignores the fact that TCP connections are the result of user and application actions. For example, the TCP connections generated from downloading a Web page, which constitute a “Web session”, are the result of a user entering a URL at a web browser or clicking on a link. A single click from a user can create more than one connection to download the embedded objects in a web-page. These connections together constitute one session and they can have different source but the same destination IP address. Such session arrivals can be consistent with an open-loop model, i.e., users generate new sessions, independent of what happened to their previous sessions and whether the network is congested or not. In other words, even though the transport layer provides congestion responsiveness through TCP, the session layer can be completely unresponsive if it keeps generating new sessions even when the network is congested.

We do not claim that TCP congestion control is not necessary. It is not sufficient, however, to avoid persistent overload. To understand this point, consider the previous example of an open-loop session layer. When the network becomes congested, each active TCP connection backs-off either reducing its send-window by a large factor or getting into a

Figure 60: The TCP feedback loop at the transport layer cannot avoid persistent overload if there is no session layer congestion control.

108
relatively long silence period (retransmission timeout). This means that congestion control pushes the offered load from each connection back to the TCP buffer of the sender. That connection is still active, however, and so it will keep trying to retransmit any lost packets and to increase its window. As the session layer keeps generating new transfers, the number of competing flows will increase, leading to a diminishing per-session goodput. TCP cannot avoid the emerging persistent overload. Instead, we need a way to tell the session (or application) layer to slow down or stop generating new flows for a while, thus forming a closed-loop system where the arrival of new flows is delayed upon congestion. Presently, this session-layer feedback is sometimes built in the application, or it simply results from the way people react when their applications are slow.

7.2 CTR Measurements

In this section, we propose a methodology to estimate the Closed-loop Traffic Ratio. The outline of the CTR estimation methodology is as follows. First, we partition the packet trace into a set of sessions initiated by each user. This process is far from simple, and it requires some knowledge of the corresponding application. For example, in the case of HTTP downloads, a “user” is associated with a specific IP destination address. Each session corresponds to a download operation requested by a user and it can consist of multiple “transfers”. After we have transformed the packet trace into a “session trace”, we then classify each session as open-loop or closed-loop. We assume that the dependency between session arrivals from the same user is related to the arrival time of the new session with respect to the finish time of the previous session from that user. Specifically, a session from a user is considered dependent on her previous session, and is classified as closed-loop, if it starts soon after the completion of the previous session. If the next session from a user starts while the previous session is still active, or after a long time from the completion of the previous session, then we view that new session as independent of the previous session and classify it as open-loop.
7.2.1 Definitions and Methodology

We start with a more detailed explanation of the key terms and of the CTR estimation methodology for the case of HTTP/HTTPS downloads. HTTP is not the only protocol/application for which we perform the following analysis. Traffic with other well-known ports is also well understood, in terms of who is the “user”, what constitutes a “session”, etc, and so we also apply the same methodology for those applications. Unfortunately, a large part of Internet traffic today does not use well-known ports. That traffic is probably generated by peer-to-peer applications. Eventually, we were able to analyze more than 50% of the TCP traffic in half of the traces we analyzed. In some traces the fraction of traffic we could analyze was as high as 78%, but in others was as low as 26%.

7.2.1.1 Users and Sessions

User is an entity (typically a person, but it can also be an automated process) that issues Web requests. Each such application-layer request is a session. We assume that a user is identified in the packet trace by a destination IP address. An important exception to this rule is when multiple users share the same host (e.g., remote login) or when the addresses of different users are translated somewhere in the network to the same address (e.g., NATs or proxies). We have devised a heuristic that can identify and ignore multi-user hosts, described in section § 7.3.

In HTTP/HTTPS, a user is associated with a destination address, because users typically download traffic. In applications that mostly upload traffic to remote hosts, the user is associated with a source address. A download session, associated with a certain user, can contact a number of different servers, and it can consist of several TCP connections with different destination ports. Consequently, the traffic that belongs to a certain session would have the same IP destination address, but potentially different source addresses and/or destination ports.
7.2.1.2 Connections and Transfers

A TCP connection is identified in the packet trace by a unique 5-tuple field (Source IP address, Destination IP address, Source Port, Destination Port and Protocol). A connection has a start and a finish time, corresponding to the timestamps of the first and last packets in the connection, respectively. We ignore connections that were ongoing at the start or end of the trace. Pure ACKs are packets without payload. A connection with more pure ACKs than data segments is considered an ACK flow and it is ignored from the analysis.

HTTP 1.1 introduced persistent connections, meaning that a connection can stay alive for a long time, transferring Web objects that belong to different sessions. We partition a connection into one or more transfers, with different transfers being part of different sessions. Figure 61 shows an example of a persistent connection that includes two transfers. The packet interarrivals within the same transfer are determined by TCP (e.g., self-clocking, retransmission timeouts) or network delays. The packet interarrivals between different transfers, however, are typically determined by the latency of user actions (e.g., clicking at a Web link). Consequently, the inter-transfer interarrivals (“gaps”) are usually much longer than the intra-transfer interarrivals. We use this observation to partition a connection into transfers. If a packet interarrival within the same connection is larger than a certain Silence Threshold (STH), which represents the maximum intra-transfer gap, then a new transfer starts with that packet. To choose a reasonable value for STH, we examined values in the range 1sec-1min. A very small STH would partition a transfer in smaller chunks, while a very large STH would merge different transfers together. So, we expect that the average transfer size increases with STH. More importantly, we expect that for a certain range of this threshold, when it falls between the larger intra-transfer gaps and the lower inter-transfer gaps, the number of transfers is almost constant. The distribution of transfer sizes as a function of STH for a Georgia Tech inbound packet trace shows that the median transfer...
size is roughly constant when the threshold is between 35-45sec. In the following, we set STH to 40sec.

7.2.1.3 Grouping Transfers into Sessions

Since a session can consist of several transfers, as shown in Figure 62, we need to identify the transfers (TCP connections or segments of TCP connections) that were generated as a result of the same user action. The key observation here is that the interarrival of two transfers that belong to the same session will typically be much lower than the interarrival of transfers that belong to two different sessions. The latter are separated by the latency of a user action. We expect transfers of different sessions to start at least one second or so from each other, while transfers that belong to the same session are typically generated automatically by the Web browser within tens or hundreds of milliseconds.

Specifically, if the interarrival between two successive transfers is larger than a certain parameter, referred to as Minimum Session Interarrival (MSI), then the second transfer starts a new session. We examine the robustness of the CTR estimate to the exact choice of the MSI in Section 7.2.2.

![Figure 62: Timeline of two sessions. Each session consists of several transfers.](image)

7.2.1.4 Classifying Sessions as Open-loop or Closed-loop

After having transformed the packet trace into a “session trace”, we classify each session as open-loop or closed-loop. Recall that the key difference between these two is that in the open-loop model sessions arrive independent of the progress of previous sessions from the same user.

The first session from a user is considered open-loop, given that that session has no dependencies to previous sessions. If that user generates a new session after her previous
session finishes and no later than the *Maximum Think Time* (MTT), then the arrival of this new session is considered dependent on the progress of the previous session. So, we classify that session as closed-loop. If, however, the new session arrived during her previous session or much later, more than MTT, we assume that the user either does not wait for her previous session’s completion, or she was inactive for some time and now returns to the network without any “memory” of previous sessions. Thus, we classify that session as open-loop. These different cases are shown in Figure 63. We examine the robustness of the CTR estimate to the exact choice of the MTT in Section 7.2.2.

![Diagram showing classification of sessions as open-loop or closed-loop based on MTT](image)

**Figure 63:** Classification of different sessions from the same user as open-loop or closed-loop.

### 7.2.1.5 Other Traffic with Well-known Ports

For other traffic, not generated by HTTP/HTTPS, we followed the convention that the transfer is an upload if the destination port is well-known, and a download if the source port is well-known. The rest of the estimation methodology is the same as in Web traffic.

### 7.2.1.6 CTR Calculation

Once we have classified sessions as open-loop or closed-loop, we then calculate the CTR as the fraction of bytes from closed-loop sessions. If the CTR is close to zero we expect that most traffic is congestion unresponsive (open-loop model), while if the CTR is close to one we expect that most traffic would reduce its flow arrival rate upon congestion (closed-loop model).

### 7.2.1.7 Limitations

The main assumption in the previous methodology is that the timing between the arrivals of successive sessions from the same user can reveal whether sessions arrive independently of each other. This is not always the case of course. For example, a user can obtain a
URL through an ongoing web-page download (session X) and then start downloading that URL in another window (session Y), while session X is still active. In this case, the arrival of session Y depends on the progress of session X. However, session Y starts before X completes, and so we would classify the former as open-loop. On the other hand, in a fast and uncongested network, even though a user can start a sequence of independent sessions, some of the sessions can complete before a subsequent session starts. In this case, the latter will be incorrectly classified as a closed-loop session. We expect that such “deviations” are not common compared to the more common Web browsing behavior, which is to first download a page (open-loop session arrival), spend some time reading or viewing it, and then either download another page (closed-loop session arrival) or leave the system for a while.

7.2.2 Robustness of Estimation Methodology

The CTR estimate depends on the following parameters: STH, MSI, and MTT. We have already mentioned that a robust value for STH is around 40sec. In this section, we investigate the optimal range for MSI and MTT, and examine the CTR’s robustness to the choice of these two parameters.

The MSI is used to merge together different transfers of the same session. As these transfers are typically machine-generated, they start almost simultaneously. In the packet trace, however, they can appear with slightly longer spacings due to network delays. A reasonable range for this parameter is between 0.5-1sec. The MTT, on the other hand, represents the longest “thinking” time for a user during Web browsing, before we assume that that user left the system. A reasonable range for this parameter is between 5-30min.

We examined the CTR variations for a Georgia Tech trace with different MSI and MTT values. The main finding is that the CTR does not significantly depend on these parameters (it varies in a small range between 0.6-0.72) as long as the MSI and the MTT fall between 0.5-2sec and 5-25min, respectively. To further examine the robustness of CTR against these two thresholds, we also performed two-factor analysis of variance. The null hypothesis is that the CTR is independent of these two parameters. We can reject this hypothesis at
a significance level of 0.05. However, the hypothesis that the CTR is independent of the interaction of these two parameters cannot be rejected at the same significance level. We found that the slope of the CTR with respect to MSI is 0.0232/sec, and with respect to MTT it is 0.0020/min. Therefore, we concluded that the CTR is practically insensitive to these parameters in the ranges that we consider. In the rest of the analysis, we set MSI=1sec and MTT=15min.

Table 2: CTR of the analyzed TCP traffic at various links.

<table>
<thead>
<tr>
<th>Trace ID</th>
<th>Direction</th>
<th>Collection time</th>
<th>Link location</th>
<th>Duration</th>
<th>GB (%)</th>
<th>bytes</th>
<th>CTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaTech-in</td>
<td>In</td>
<td>07-Jan-05</td>
<td>GaTech</td>
<td>2Hr.</td>
<td>129.74</td>
<td>63.50%</td>
<td>0.71</td>
</tr>
<tr>
<td>GaTech-out</td>
<td>Out</td>
<td>07-Jan-05</td>
<td>GaTech</td>
<td>2Hr.</td>
<td>208.06</td>
<td>47.90%</td>
<td>0.57</td>
</tr>
<tr>
<td>Los-Nettos</td>
<td>Core</td>
<td>03-Feb-04</td>
<td>Los-Nettos, CA</td>
<td>1Hr.</td>
<td>59.37</td>
<td>65.59%</td>
<td>0.77</td>
</tr>
<tr>
<td>UNC_em0</td>
<td>Out</td>
<td>29-Apr-03</td>
<td>UNC</td>
<td>1Hr.</td>
<td>153.19</td>
<td>35.76%</td>
<td>0.61</td>
</tr>
<tr>
<td>UNC_em1</td>
<td>In</td>
<td>29-Apr-03</td>
<td>UNC</td>
<td>1Hr.</td>
<td>41.51</td>
<td>26.59%</td>
<td>0.76</td>
</tr>
<tr>
<td>UNC_em1-2</td>
<td>In</td>
<td>24-Apr-03</td>
<td>UNC</td>
<td>1Hr.</td>
<td>55.25</td>
<td>44.88%</td>
<td>0.78</td>
</tr>
<tr>
<td>MFN_0</td>
<td>Core</td>
<td>14-Aug-02</td>
<td>MFN, San Jose</td>
<td>1Hr.</td>
<td>151.38</td>
<td>61.09%</td>
<td>0.69</td>
</tr>
<tr>
<td>MFN_1</td>
<td>Core</td>
<td>14-Aug-02</td>
<td>MFN, San Jose</td>
<td>1Hr.</td>
<td>186.93</td>
<td>71.83%</td>
<td>0.62</td>
</tr>
<tr>
<td>IPLS_0</td>
<td>Core</td>
<td>14-Aug-02</td>
<td>Abilene</td>
<td>1Hr.</td>
<td>172.22</td>
<td>41.93%</td>
<td>0.60</td>
</tr>
<tr>
<td>IPLS_1</td>
<td>Core</td>
<td>14-Aug-02</td>
<td>Abilene</td>
<td>1Hr.</td>
<td>177.99</td>
<td>47.27%</td>
<td>0.64</td>
</tr>
<tr>
<td>Auckland_0</td>
<td>In</td>
<td>06-Nov-01</td>
<td>Auckland, NZ</td>
<td>6Hr.</td>
<td>0.58</td>
<td>72.99%</td>
<td>0.73</td>
</tr>
<tr>
<td>Auckland_1</td>
<td>Out</td>
<td>06-Nov-01</td>
<td>Auckland, NZ</td>
<td>6Hr.</td>
<td>1.44</td>
<td>77.99%</td>
<td>0.67</td>
</tr>
</tbody>
</table>

7.2.3 Results

Table 2 summarizes the metadata of the traces we analyzed here. The traces are from university access links, commercial access links and backbone links, and they were collected between 2001-2005. It also shows the CTR estimates for Web and other well-known port traffic in these 12 Internet traces.

An important observation is that the CTR for access links is always higher in the inbound direction than in the outbound direction. This can be explained by the fact that users that initiate sessions in the inbound direction belong to the limited population of users within that campus network. On the other hand, users that initiate sessions in the outbound direction come from all over the Internet and they belong to a much larger population. Consequently, the fraction of open-loop traffic in the latter is higher (lower CTR).

The most important observation, however, is that the CTR for almost all traces is high, typically between 60-80%. Even the backbone links, where we would expect more open-loop traffic due to the large number of users, have a high CTR. This suggests that a major reason
for the congestion responsiveness of Internet traffic may be that most applications follow the closed-loop model, and so they are responsive to congestion at the session generation layer.

### 7.3 Multi-user Host Detection

We describe a heuristic to distinguish between single-user and multi-user hosts (such as NATs, proxies, rlogin servers etc). A host is identified by a unique IP address in the packet trace. In a multi-user host, sessions generated by different users share the same destination address, making it impossible to distinguish sessions from different users. In multi-user hosts, however, the number of transfers per session would typically be much larger than in single-user hosts. This large difference is the key criterion to detect multi-user hosts. For instance, in one of the Georgia Tech inbound packet traces about 99% of hosts have less than 20 transfers per session. However, there are also 100 hosts (<1%) that generate up to a few thousands of transfers per session; it is likely that they are multi-user hosts. We examined the DNS names of those hosts, and several of them indicate proxies and firewalls. So we chose a threshold of 10 transfers per session, on the average, to distinguish single-user from multi-user hosts. It turns out that the final CTR estimate (which is what we care about) is robust to the previous threshold, as long as the latter is more than 5-6 and less than about 100 sessions.

### 7.4 Summary

This work focused on a simple question: how does Internet traffic react to congestion? This is an important question towards a better understanding of the Internet traffic characteristics. Additionally, the issue of congestion responsiveness has several practical implications. To avoid overload conditions, applications need to be “congestion-aware” at the session layer. One way to do so is to delay generating a new session to a server if the previous has not yet completed or if it is stalling. If the traffic at a certain link is mostly open-loop, then the operator has two options. One is to increase the link capacity above the given offered load. The second is to do some sort of admission control at that link. With closed-loop traffic, on the other hand, these two options may not be necessary because the offered load is
self-regulating. We are currently extending this work to estimate the CTR for peer-to-peer traffic, which often accounts for most of the traffic today.
CHAPTER VIII

CONTRIBUTIONS AND FUTURE WORK

8.1 Research Contributions

TCP is often blamed for poor end-to-end performance in the Internet. The emergence of high-speed networks in recent years has highlighted the inefficiencies of TCP. While many proposals for improving/replacing of TCP have been made, little agreement has been reached towards a suitable replacement. In this thesis, we ask a slightly different question, that is, can we improve end-to-end performance without replacing current TCP? We study different factors that affect end-to-end performance and may still be relevant in the coming generations of the Internet.

In this thesis, we identified that the current TCP’s blind probing for more bandwidth causes self-induced losses in uncongested paths. Such losses result in both poor end-to-end performance and wasted link capacity. We showed the need of distinguishing between self-induced losses and congestive losses, emphasizing that the former can be avoided, while the latter can not. We developed an application layer mechanism SOBAS. SOBAS infers if it is operating in an uncongested path and sets its socket buffer size so that it does not seek more bandwidth than what is available. We have also shown that in a congested path there is no incentive for a TCP transfer to limit its socket buffer size. Therefore, if SOBAS realizes that it’s in a congested path, it leaves the socket buffer size to the maximum allowable value and lets TCP congestion control decide its fair-share.

We investigated the problem of router buffer sizing, focusing on the end-to-end performance. We showed that the ratio of the input to the output capacity determines the required buffer size at any router interface. This result brings out a surprising fact that the buffer requirement at any router can be substantially different in two directions depending on the capacities served on those sides of the router.

With the help of the open-loop and the closed-loop models of traffic generation, we have
shown the importance of session/application layer congestion control. Lack of such control can result in congestion unresponsive traffic aggregates even if all the constituent flows react to congestion using TCP congestion control algorithms. We showed that the closed-loop flow arrival process leads to a congestion responsive traffic. We also presented a passive analysis scheme that classifies traffic in a link as following an open-loop or a closed-loop model. Such classification can be used to quantify congestion responsiveness of the traffic aggregate in that link.

8.2 Future Directions

In the current SOBAS implementation, one of the assumption is that the cross-traffic remains stationary for the duration of the transfer. This assumption is valid if the TCP transfer lasts for short duration (up to few minutes). Longer TCP transfers, however, can face non-stationary cross-traffic conditions. In such conditions, SOBAS may either fail to capture extra bandwidth made available by departing flows or fail to notice a reduced available bandwidth due to the start of a new flow and thus cause self-induced losses. One possible way to make SOBAS more efficient in non-stationary conditions could be to add some light weight mechanism to detect changes in the background traffic and adjust the socket buffer size accordingly.

In this thesis, we studied router buffer sizing that maximizes TCP throughput. We assume that the buffer sizing decisions are made locally, considering one router at a time. If a network operator wants to somehow optimize all router buffers in the network, he/she would need to consider the effects of buffer sizing on the characteristics of the traffic that will arrive at downstream routers. Such network-wide optimizations are beyond the scope of this thesis.

Router buffer sizing can also be performed for other application-oriented performance metrics, such as, HTTP transfer latency, VoIP and interactive video quality, video streaming quality, gaming as well as other interactive applications.

In this thesis, we classify non-persistent transfer using either the open-loop or the closed-loop models. While these models are simple, tractable, and provide some key insights for
the aggregate traffic behavior, they are not very accurate models for actual user behavior. More detailed user behavior models need to be developed in order to design better tools to predict input traffic characteristics and their response to overload condition. With the popularity of youtube and other video streaming applications, we now experience a surge in applications that transfer a large amount of data for each user action of entering an URL or following a link. The impact of this shift in the session size distribution is yet to be understood.

In order to classify a session as part of an open-loop or a closed-loop process, one needs to determine whether the session was initiated by the sender or the receiver. In this work, we classify traffic from or to well-known ports, as we can determine the initiating end of these flows. However, there is a significant amount of traffic that does not use well-known ports. Determining whether the source or the destination initiates these flows remains an open problem and need to be addressed before one can classify that part of the traffic.
REFERENCES


VITA

Ravi Prasad received the Bachelor of Technology degree in Ocean Engineering and Naval Architecture from Indian Institute of Technology, Kharagpur in 1998 and Master of Science degree in Civil Engineering from University of Delaware, Newark in 2001. He started his graduate studies in Computer Science at the University of Delaware under guidance of Prof. Constatine Dovrolis, before joining the College of Computing at Georgia Institute of Technology, Atlanta. His current research interests include router buffer sizing, bandwidth estimation methodologies, network measurements and applications, and TCP in high bandwidth networks.