On the impact of variability on the buffer dynamics in IP networks *

Youngmi Joo, Vinay Ribeiro,
Anja Feldmann, Anna C. Gilbert, and Walter Willinger

Abstract

The main objective of this paper is to demonstrate in the context of a simple TCP/IP-based network that depending on the underlying assumptions about the inherent nature of the variability of network traffic, very different conclusions can be derived for a number of well-studied and apparently well-understood problems in the areas of traffic engineering and management. For example, by either fully ignoring or explicitly accounting for the empirically observed variability of network traffic at the source level, we provide detailed *ns-2*-based simulation results for two commonly-used traffic workload scenarios that can give rise to fundamentally different buffer dynamics in IP routers. We also discuss a set of *ns-2* simulation experiments to illustrate that the queueing dynamics within IP routers can be qualitatively very different depending on whether the observed variability of measured network traffic over small time scales is assumed to be in part endogenous in nature (i.e., due to TCP's feedback flow control mechanism, which is "closed loop") or is exogenously determined, resulting in an "open loop" characterization of network traffic arriving at the routers.

1 Introduction

Traffic characterization and modeling are generally viewed as important first steps toward understanding and solving network performance-related problems. At the same time, there is little disagreement that the resulting understanding of and solutions to network performance problems are only as good and complete as the underlying assumptions on the usage of the network and the nature of the traffic that it carries. The main goal of this paper is to highlight with a toy example of a TCP/IP network the extent to which assumptions underlying the nature of network traffic can influence practical engineering decisions. More specifically, using the *ns-2* network simulator [1], we illustrate how by either implicitly accounting for or explicitly ignoring the empirically observed variability of network traffic over large and small time scales, a range of different and at times opposing conclusions can be drawn about the inferred buffer dynamics for IP routers. While there are many known causes for the observed variability in measured TCP traffic (e.g., see [4]), in this paper we focus on just two of them. On the one hand, we consider a known cause for variability of the packet rate process over large time scales; that is, high variability at the connection level resulting in self-similar scaling. On the other hand, we investigate a suspected cause for variability of the rate process over small time

*Y. Joo is with the Department of Electrical Engineering, Stanford University, CA 94305-9030, email: jym@leland.stanford.edu. V. Ribeiro is with the ECE Department, Rice University, Houston, TX 77005, email: vinay@rice.edu. A. Feldmann, A. Gilbert, and W. Willinger are with AT&T Labs-Research, Florham Park, NJ 07932-0971, email: {anja, agilbert, walter}@research.att.com.

scales; that is, the dynamics or actions of TCP resulting in complex scaling behavior of the traffic over small time scale that is consistent with multifractal scaling [5].

As far as large time scale variability is concerned, the extreme case of no variability can be achieved by requiring at the application layer that each active source has to transfer an essentially infinite file for the duration of the entire simulation. This *infinite source model* lacks the natural variability that has been observed in measured data at the application level and, in turn, lacks any non-trivial large time scale variability in the corresponding rate process [11, 3, 6]. In contrast, large time scale variability that is consistent with self-similar scaling over those time scales and has become a trademark of measured traffic rate processes can be achieved in a parsimonious manner by explicitly accounting for an adequate level of variability at the application layer [15, 14]. In fact, by replacing the infinite source model by a SURGE-like workload generator [2], that is, by changing an infinite file transfer into an application that imitates a typical Web session (with appropriately chosen heavy-tailed distributions to match the observed variability of various Web-related items such as session length and size, size of requested Web pages), we obtain a *Web-user source model* that automatically generates the desired large time scale variability in the aggregate rate process. These workload models are described in more details in Section 2.

Subsequently, in a first set of *ns-2* simulation experiments, we demonstrate in Section 3 how the two source models lead to two qualitatively very different queueing dynamics in the router. While the queueing behavior with the infinite source models is completely dominated by synchronization effects, these effects essentially disappear when the sources are Web users. Using the terminology originally due to V. Jacobson, we illustrate how the presence of many short-lived TCP connections or "mice" – a salient feature of our Web workload model – completely changes the buffer dynamics that has been noted in the past when the network load is entirely due to the long-lived TCP connections or "elephants" of the infinite source model. For earlier studies on TCP dynamics assuming infinite source models, see for example [9, 12, 16, 17]).

In a second set of *ns-2* simulation experiments, we focus in Section 4 on the small time scale variability of the packet rate process and its likely cause, that is, the feedback flow control mechanism that is part of the TCP/IP protocol suite [5, 4]. In particular, we summarize the results of our studies on how the network through the TCP end-to-end congestion control algorithm shapes (with some delay) the packet flow emitted from the different sources, which in turn alters the rate process that arrives at the IP router for buffering (which in turn impacts the level of congestion, etc.). The results are obtained by performing a number of related *closed loop* and *open loop* simulations and comparing them on the basis of some commonly-used performance criteria. Here, by "closed loop" we mean a *ns-2* simulation with a fixed networking configuration, including buffer size in the router(s), link bandwidths, delays, etc., and where all hosts use TCP. In contrast, "open loop" means we collect a packet trace from a particular *ns-2* simulation run (or alternatively, from a link within the Internet) and use it to perform a trace-driven simulation of a queueing system that represents our IP router. Note that trace-driven simulations cannot account for the capabilities of the network to shape and thus alter the offered traffic to the queue.

We conclude in Section 5 by commenting on a number of admittedly unrealistic assumptions regarding our simulation configuration. However, despite these over-simplifications and a number of other shortcomings of our studies, we believe that the findings reported in this paper will increase the overall awareness that it is in general easy to draw conclusions based on infinite source models and/or open loop systems and their performance, but that the real challenges lie in convincingly demonstrating that these conclusion either still hold or become invalid for realistically loaded networks and/or the corresponding closed loop systems and their performance.

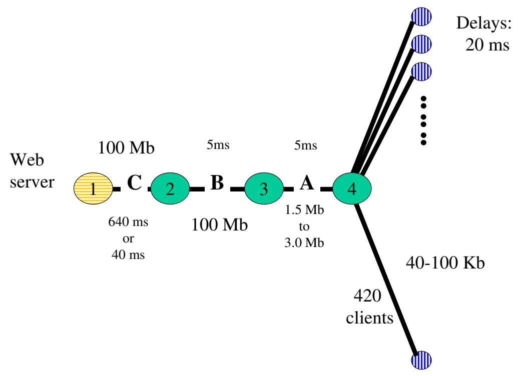


Figure 1: Network configuration

2 The Simulation Setup

In this section, we give a description of the networking configuration used throughout this paper and spell out the details of our workload models. In addition, we discuss the simulation engine used for our studies and review some TCP-specific features that are necessary to present and discuss the findings of our simulation experiments.

2.1 A simple networking configuration

All of the simulation experiments reported in this paper involve the simple network topology depicted in Figure 1. Our toy network consists of a single server (node 1), a set of low-speed clients (nodes $5 - 405$), and a number of links. The clients are connected to the network via $40 - 100$ Kbps links, the server has a 100 Mbps connection to the network, and two additional links (A, B) comprise the rest of the network. Link A is used to limit the capacity of the network to a value between 1.5 Mbps and 3 Mbps and represents, in fact, the bottleneck link in our simulation setup. The link B bandwidth is set at 100 Mbps, and we use this link to observe the dynamics of the traffic before it traverses the bottleneck link A. To gauge the impact of the buffer size in the router, we vary the number of buffer spaces available for buffering packets[1] in node 3 for link A; depending on the particular experiment, this number can vary anywhere from 10 to 1000. Note that this network configuration allows for no variability as far as delays, round-trip times and cross traffic are concerned. While all of these three aspects of realistic network traffic are known to be crucial for matching the variability inherent in measured Internet traffic [4], our focus in this paper is on an extreme case of networking homogeneity, where there is a single bottleneck through which all the traffic has to go and where all packets experience one and the same delay in the network, namely 1.4 seconds (to compare, we also experiment with a link delay of 0.14 seconds). We chose this admittedly unrealistic network configuration because we wanted to illustrate our findings in a setting that has been used in a number of previous studies of the dynamics of TCP (e.g., [12, 16, 17]).

2.2 Two different workload models

In contrast to our homogeneous network configuration, our workload models for the clients are heterogeneous in nature and allow for a spectrum of variability at the user level. On the one side of the spectrum, we deal with the case of *no* user level variability by considering 50 *infinite sources* that always have data to transfer for the duration of the entire simulation (that is, 4200

[1]Ns-2 allocates buffer space in terms of number of packets and not number of bytes.

Name	number	inter–page	objects/page	inter–object	object size
INFINITE SOURCE	Constant 50	Constant 1	Constant 1	—	Constant 1000000
WEB SOURCE	Constant 350	Pareto mean 50 shape 2	Pareto mean 4 shape 1.2	Pareto mean 0.5 shape 1.5	Pareto mean 12 shape 1.2

Table 1: Summary of the relevant distributions (with parameter values) for the two workload used in our simulation experiments.

seconds). To eliminate any transient effects, the sources start their transfers at random times – picked according to a uniform distribution – during the first 600 seconds of the simulation, and when analyzing the output of our simulations, we focus on the remaining 3600 seconds of each simulation run when all 50 source are known to be active.

To cover the other side of the spectrum, we rely on a *Web workload* model considered in [4] that is very similar to SURGE developed at Boston University [2]. The main idea behind these Web workload models is that during a Web session, a user typically requests several Web pages, where each Web page may contain several Web objects (e.g. `jpg` images or `au` files). To parsimoniously capture the observed variability within the different layer of this natural hierarchical structure of a typical Web session (e.g., see [3, 6, 2]), we consider certain types of probability distribution functions for the following session attributes: number of pages per session, inter-page time, number of objects per page, inter-object time, and object size (in KB). The specifics of these distributions, including the corresponding parameter values, are given in Table 1. Note that the empirically observed high variability associated with these Web Session attributes is naturally captured via Pareto-type distributions with appropriately chosen parameter values.

In practice, a Web session will request a set of pages (usually 300) in the following manner.[2] After the completion of the download of all objects in the last page, it will wait for a random amount of time (sampled from the inter-page time distribution) before requesting the download of the next Web page. At the start of the next Web page download, the client determines the server, which in our setup is always node 1. In addition, the client chooses the number of objects in the next Web page by picking a random number according to the objects-per-page distribution. For each object, the client determines the size of the object (by sampling from the object size distribution) and then sends a request to the server to download the object. The time between the request for two different objects within a Web page is chosen according to the inter-object-time distribution. Once all objects of a Web page have been downloaded, the process repeats itself, i.e., after a waiting time (sampled from the the inter-page-time distribution), the next Web page is downloaded, etc.

In our simulations, we always use at least 300 Web sessions, all of which start at a random point in time within the first 600 seconds of each simulation run. Also note that the infinite source model can be viewed as a special case of our Web workload model, where the number of pages per client is 1, the number of objects per page is 1, and the object size is set to a very large value (e.g., 1000000) to ensure that the client does not run out of data to send for the duration of the simulation. In this sense, Table 1 provides a complete specification of the two workload models used in our experiments below.

[2]Note that in a typical HTTP 1.0 transaction, a Web client sends a request to the Web server for a Web object after establishing a TCP connection. The Web server responds with a reply header and then continues to send the data. To circumvent some limitations in the original *ns-2* TCP connection module we emulated the exchange of the HTTP header information with two TCP connections.

2.3 Some TCP background

The simulation engine used throughout this study is *ns-2* (Network Simulator version 2) [1]. This discrete event simulator provides a rich library of modules such as UDP, different flavors of TCP, scheduling algorithms, routing mechanism, and trace collection support. For the purposes of this paper, we rely on the fact that *ns-2* comes with a thoroughly-checked and well-studied implementation of TCP. To fully appreciate this advantage of using *ns-2* for our experiments, we first review those features of TCP which are of particular interest for the purpose of this study. For a more comprehensive treatment of TCP, see for example [13].

TCP is one of the predominant transport layer protocols in the Internet, providing a connection-oriented, reliable, byte stream between a source (or sender) and a destination (or receiver). TCP provides reliability by splitting the data into numbered segments. To insure that each segment is transmitted reliably, TCP – among other precautions – maintains a timer, RTO, for the different segments. If no acknowledgment for the segment is received by the sender from the receiver within the timer period RTO, TCP assumes that the segment has been lost and retransmits it. Another way for TCP to detect losses is based upon duplicated acknowledgments. Since acknowledgments are cumulative, and since every segment that is received out of order (i.e., non-consecutive segment numbers) triggers an acknowledgment, TCP assumes that four duplicated acknowledgments indicate that a segment was lost. As before, TCP in this case retransmits a segment if it detects a lost segment.

Since each end-system has limited buffer space and the network has limited bandwidth, TCP provides end-to-end flow control using a sliding window protocol. As far as the buffer-limited end-systems are concerned, the sender computes a usable window which indicates how much data the receiver buffer can handle. To deal with the bandwidth-limited network, the sender maintains a congestion window of size CWND. The sender can transmit up to the minimum of CWND and the usable window. At the start of a TCP connection, the usable window is initialized to the buffer size of the receiver, and the congestion window is limited to one or two segments. Each received acknowledgment, unless it is a duplicate acknowledgment, is used as an indication that data has been transmitted successfully and allows TCP to move the usable window and to increase the congestion window. However, increasing the congestion window depends on the state of the TCP connection. If the connection is in slow start, it is increased exponentially, or more precisely, by one segment for every acknowledgment. If the connection is in congestion avoidance, it is increased linearly by the maximum of one segment per round-trip time. TCP switches from slow start to congestion avoidance if the size of the congestion window is equal to the value of a variable called the slow start threshold, or SSTHRESH, for short. If TCP detects a packet loss (which in today's Internet is interpreted as an indication of congestion), either via timeout or via duplicated acknowledgments, SSTHRESH is set to half of the minimum of CWND and the usable window size. In addition, if the loss was detected via timeout, the congestion window is set to one segment.

3 Impact of variability at the application layer

In this section we demonstrate how the TCP flow control algorithm can lead to artifacts in the router buffer dynamics at node 3, depending on which of the two workload models are used to generate the network traffic. To this end, we compare the scenario where TCP only has to handle long-lived connections or "elephants" with that where it is faced with a mixture of "elephants" and short-lived "mice."

3.1 No variability and high synchronization

We first consider the case where 50 clients generate traffic according to the infinite source workload model and show in the left plot of Figure 2 some typical features associated with having 50 long-lived TCP connections or "elephants" responsible for all traffic seen on the network. More specifically, the left plot shows (i) the traffic rate process (i.e., number of packets per second) as it arrives at the queue at node 3, to be forwarded through the bottleneck link A to the different destinations, and (ii) the instantaneous buffer occupancy of the queue at node 3. We assume that the maximum buffer occupancy of the queue at node 3 is 50 packets and that the queueing discipline is "drop-tail", i.e., whenever an arriving packet sees a full buffer, it is dropped by the router. Each packet drop is viewed by TCP as an indication of network congestion and results in a signal back to the sender of the dropped packet to slow down its sending rate, which in turn reduces the overall packet rate process arriving at the node 3 queue. This feedback dynamic is an inherent feature of TCP's end-to-end flow control mechanism, and as far as the case of "elephants" is concerned, Figure 2 illustrates the effect of this dynamic on the rate process and the buffer occupancy. Shortly after the queue is filled and packets are dropped, the rate process stops increasing and drops after some time from a maximum of around 215 packets per second to about 150 packets per second. Such a drastic reduction in the rate process arriving at the queue allows the queue to completely drain its buffer. Upon experiencing no dropped packets, the individual TCP connections start to increase their sending rates again, which in turn results in an increase of the overall packet rate process (from about 150 to 215 or so packets per second) as seen by the queue. As a direct result of this higher arrival rate, the buffer at the node 3 queue starts to fill up again. Once the buffer is full, another round of packet drops will cause the affected connections to again reduce their sending rates, and the same process as before repeats itself. These arguments explain the predominant periodic fluctuations of the packet rate process and the buffer occupancy process depicted in the left plot of Figure 2, and the results are fully consistent with some of the original studies of the dynamics of TCP as described, for example, in [12] [17].

To demonstrate that these pronounced synchronization effects for the packet rate and buffer occupancy processes are not an artifact of the extremely large delay value of 640 milliseconds for link C, we show in the right plot of Figure 2 the results for of the same simulation experiment, except that the link C delay is now reduced to 40 milliseconds. Note that we still observe pronounced periodic behavior, but since the feedback to the clients is now much more immediate, the queue does not have time to drain completely, the cycle length is significantly shorter, the overall link utilization is larger, and the rate process is somewhat less variable. Nevertheless, the arguments for the observed synchronized behavior remain the same as before: many of the "elephants" experience multiple packet drops within a short period of time and are thus forced into into slow-start; from this point on, they proceed more or less in lock-step.

3.2 High variability and no synchronization

To demonstrate how allowing for realistic variability at the application level changes the qualitative features of Figure 2, we replace the infinite sources by clients that generate traffic according to our Web workload model. Intuitively, while retaining a few "elephants," this source model ensures that a significant portion of the TCP connections are short-lived (i.e., representing many "mice"). The results are shown in Figure 3, where we again depict the packet rate and buffer occupancy processes for the case of a 640 milliseconds (left plot) and 40 milliseconds (right plot) link C delay, respectively. In stark contrast to the visually obvious synchronization effects in the case of the infinite sources, the inherent variability of the TCP connections in the case of the Web sources results in a complete lack of any synchronization effects.

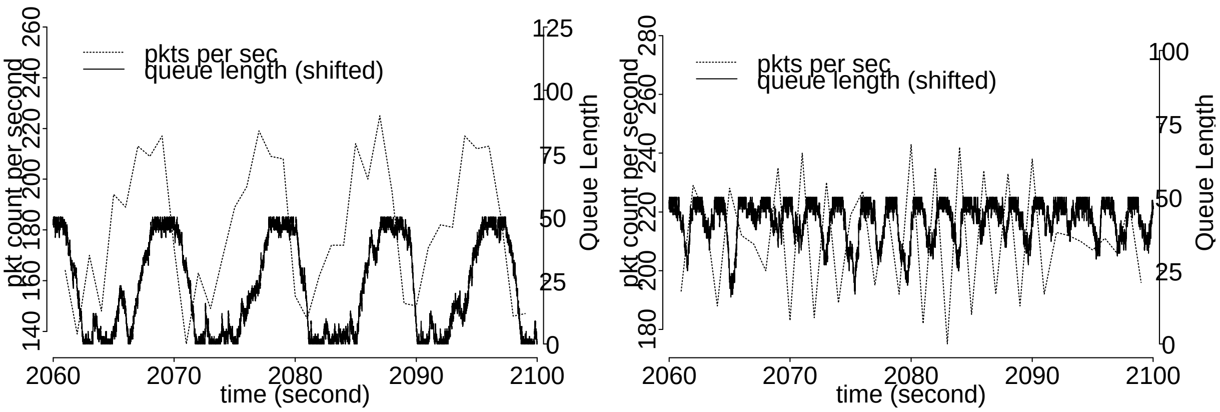


Figure 2: Packet rate process and buffer occupancy process (left: 50 infinite sources, link C delay 640 milliseconds; right: 50 infinite sources, link C delay 40 milliseconds).

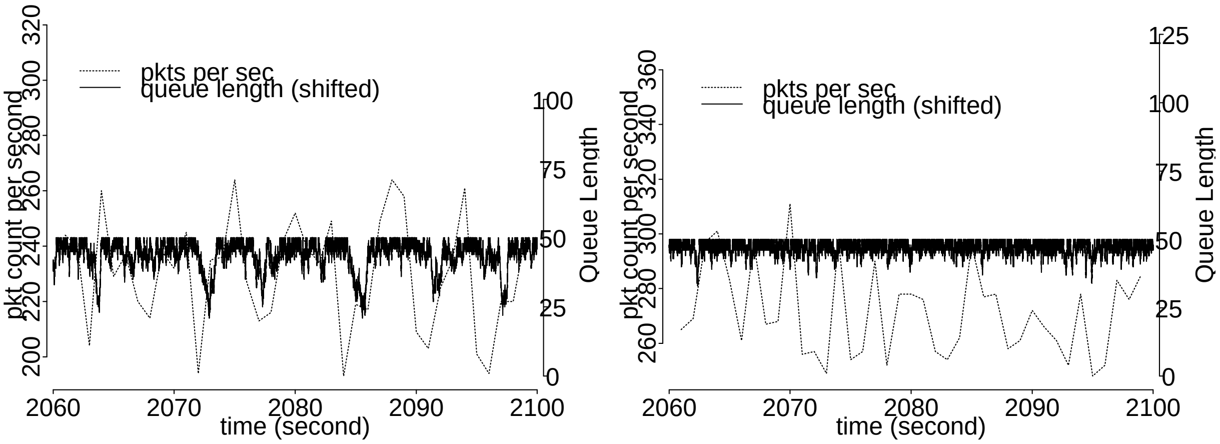


Figure 3: Packet rate process and buffer occupancy process (left: 350 PARETO 1-type Web clients, link C delay 640 milliseconds; right: 350 PARETO 1-type Web clients, link C delay 40 milliseconds).

The difference between the two workload scenarios is especially striking for the case of a link C delay of 640 milliseconds. While in the presence of many "mice", the buffer never has a chance to completely drain, it does so on a regular basis for the "elephants-only" case. Also note that while in the case of the Web sources, the packet rate process arriving at the node 3 queue never drops below 190 or so, it drops to about 140 in the case of infinite sources. Together, these effects result in a significantly higher overall utilization when there exists a proper mixture of "mice" and "elephants.". In the "elephants-only" case, even if we were to increase the number of long-lived connections, because of the presence of synchronization caused by the interaction between this type of workload and TCP feedback, the overall link utilization would not increase significantly.

3.3 On why "mice" can get rid of synchronization

In contrast to the infinite source model, our Web workload model – via its built-in Pareto-type distributions for the different Web session attributes – guarantees that a significant amount of TCP connections are very small and hence short-lived. However, TCP's feedback-based con-

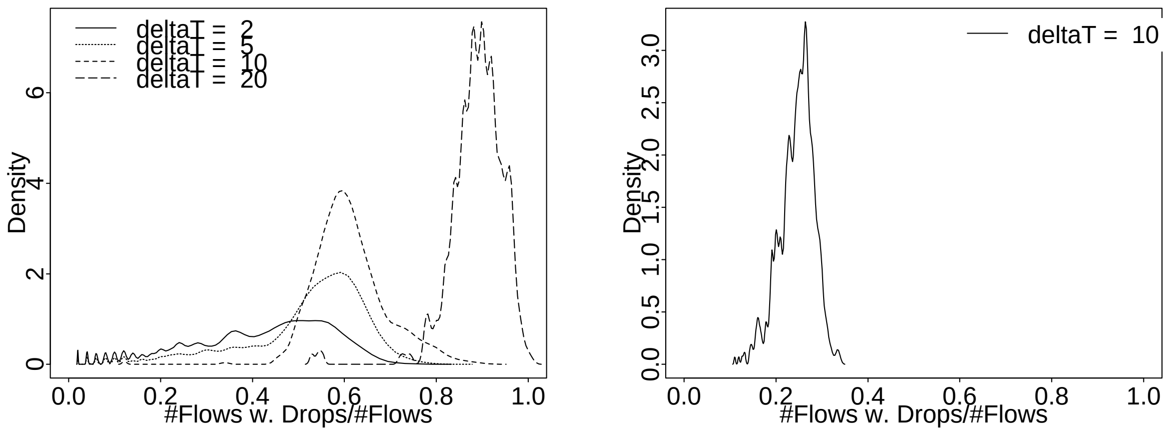


Figure 4: Effect of synchronization: Fraction of connections that experience packet losses.

gestion control mechanism was not intended for dealing with all these "mice" but was instead designed to work well in the presence of "elephants" that generate the overall network load. In fact, in the case of the "mice," it is hard or even pointless to rely on feedback because by the time the feedback signal reaches the relevant source, that source typically has no more data to sent. Although in today's Internet, the "elephants" are responsible for a major portion of the overall workload (i.e., number of bytes), the total number of packets due to the "mice" generates sufficient traffic to create losses at random points in time. This feature and the fact that the arrival patterns of the "mice" tend to be highly bursty (e.g., see [5]) suggest that the presence of significantly many "mice" makes it nearly impossible for the "elephants" to synchronize.[3]

One way to gauge the degree or severity of synchronization among the different TCP connections is to check what percentage of connections (out of all connections) experiences a loss of (at least) one packet during a time interval of size ΔT. Figure 4 shows the densities of these fractions for our two different workloads models and for different values of ΔT. Note that for the infinite source model (left plot), as ΔT increases from 1 second to 5 and 10 seconds, the densities become more centered around 60%, and when ΔT is increased even more to 20 seconds, the corresponding density function shifts toward 100%, with a mean larger than 90%. Thus, the TCP states of more than 50% of the connections can, in general, be assumed to be very similar. Moreover, given that the buffer occupancy process in Figure 2 (left plot) has a periodicity of about 10 seconds, we can conclude that about 60% of all connections lose at least one packet within this period and that almost every connection loses at least one of its packets within two such periods. Thus, even if a connection managed to avoid dropping a packet during one congestion epoch, it is almost certain to experience a packet drop during the subsequent cycle. In contrast, for the Web workload model (right plot), the density functions corresponding to the different ΔT values turn out to be essentially identical to one another (the plot only shows the density corresponding to $\Delta T = 10$), and they are all sharply concentrated around 25%. To explain this difference, note that "mice" can start when other connections are reducing their sending rates, and they often finish before they experience any dropped packet. This also explains why the fraction of connections experiencing packet drops is significantly smaller than in the absence of any "mice."

For another way to illustrate how the presence of many "mice" manifests itself in the observed TCP dynamics, we consider all connections that had (at least) one of their packets dropped and depict in Figure 5 the percentage p of those connections that were in slow-start

[3] Another known cause that works against the presence of synchronization effects in real network traffic is the observed variability in round-trip time [4], but this cause is not accounted for in our simulation setup.

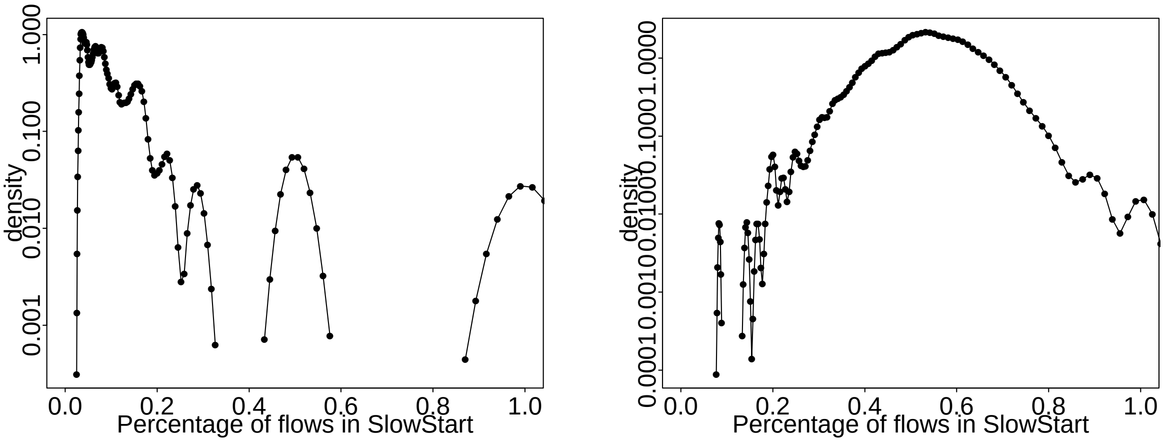


Figure 5: Effect of synchronization: Fraction of connections in slow-start.

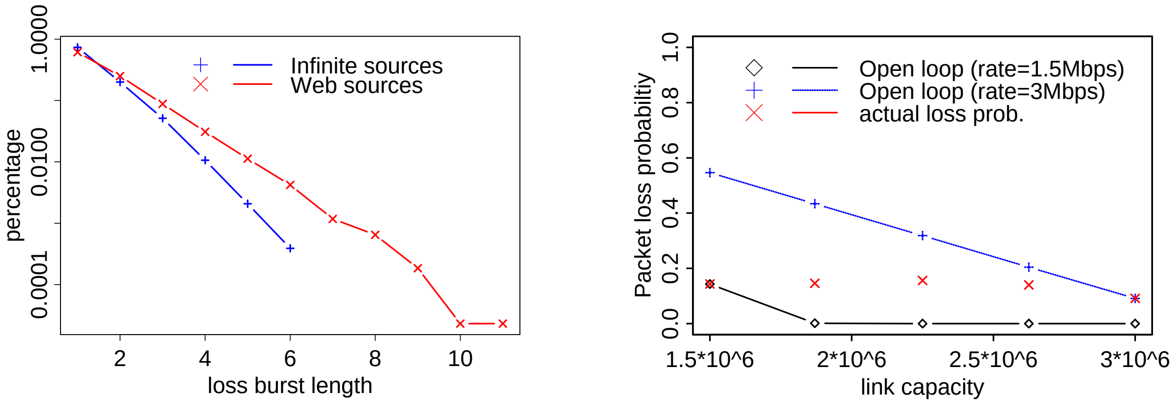


Figure 6: Left plot: Effect of synchronization – distribution of number of consecutive packet drops. Right plot: Open vs. closed loop – loss probability as a function of bottleneck link capacity.

at the time of packet drop (note that $1-p$ gives the percentage of connections in congestion avoidance); the left plot is for the infinite source model, the right plot shows the same information for the Web source model.[4] The differences between the plots are again telling. In the presence of many "mice" (right plot), most of the connections that had any packet dropped are in slow-start, meaning that they are either very short or experience multiple dropped packets; a closer look (not shown here) reveals that both cases generally contribute to this dominant slow-start effect. In any case, with many of the connections going through slow-start, thereby increasing their bandwidth usage much more aggressively than those that are in congestion avoidance, the Web sources generally succeed via the presence of the many "mice" to claim any unused bandwidth and utilize the network resources efficiently. In contrast, in the absence of any "mice" (left plot), lots of the affected connections are in congestion avoidance and increase their bandwidth usage more gradually. Indeed, if we consider (not shown here) the size of the congestion window when losses occur for the infinite source case, it turns out to be significantly larger than when the sources generate traffic according to our Web workload model.

As a result of these observations, it is reasonable to expect the dynamics of packet drops

[4]These percentages are calculated for successive time intervals of length two seconds, ignoring intervals without any dropped packets. Different choices for the length of the time interval yields essentially identical plots.

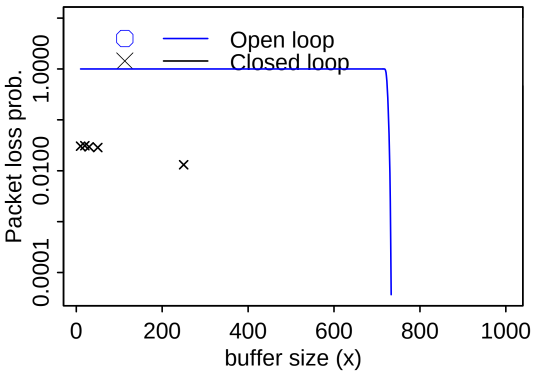
to be qualitatively different for the two workload scenarios. To confirm this conjecture, we consider in Figure 6 (left plot) the distribution of the number of consecutively dropped packet (for the aggregate packet stream) for the infinite source and Web source models. As expected, the infinite source model results in a distribution that implies a less bursty drop process than for the Web source model. To explain, "elephants" are more likely to be in congestion avoidance than in slow-start, which means that they can only send one packet for every received acknowledgment. In contrast, we have seen that the presence of many "mice" results in many connections being in slow-start, which in turn increases the likelihood that more than one packet of a given flow can be dropped within a short period of time. In other words, while the loss dynamics induced by TCP when there is no variability at the application layer results in small bursts of consecutive packet drops, these bursts can be significantly larger when allowing for application-layer variability in accordance with our Web workload model.

To summarize, our simulation experiments have demonstrated that the dynamics of TCP can interact in intricate ways with the dynamics of the underlying workload model for the sources. Static workloads such as infinite sources allow for no variability at the source level and are destined to synchronize and to proceed in lock-step through periods of no congestion and periods of congestion. However, as soon as the workload model accounts for sufficient variability at the source level, the "elephants" are forced to compete with many "mice" for the available network resources. The resulting heterogeneity in TCP states is sufficient to break up any potential synchronization effects and as a result, uses the available resources more efficiently. In contrast, synchronization generally leads to lower link utilization, less bursty losses, and more homogeneity in terms of TCP states.

4 On the impact of feedback flow control

In the following, we briefly describe some of the results of a second set of *ns-2* simulation experiments that was intended to demonstrate how the network through the TCP end-to-end congestion control algorithm shapes (with some delay) the packet flow emitted from the different sources, which in turn alters the rate process that arrives at the IP router for buffering (which in turn impacts the level of congestion, etc.). To this end, we performed a number related *closed loop* and *open loop* simulations and compared them on the basis of some commonly-used performance criteria. Here, by "closed loop" we mean a *ns-2* simulation with a fixed simple topology, including buffer size in the router(s), link bandwidths, delays, etc. and where all hosts use TCP. In contrast, "open loop" means we collect a packet trace from a particular *ns-2* simulation run and use it to perform a trace-driven simulation of a queueing system that represents our IP router. Note that the open loop nature of trace-driven simulations cannot account for the capabilities of the network to shape and thus alter the offered traffic to the queue (e.g., as a result of changing congestion levels in the network through, say, increasing the buffer in the router or by means of changing the capacity of the bottleneck link).

To illustrate, a commonly used method for investigating the buffer dynamics in a queue with constant service time is to consider an actual packet-level traffic trace, collected from some link in, say, the Internet, and use it as input for a number of open loop trace-driven simulations. For example, by changing the bandwidth of the output link of the queue, we can change the service or drain rate of the queue and, as result, study the packet loss probability as a function of the (bottleneck) link bandwidth. The right plot in Figure 6 shows the results of (i) a set of 5 open loop trace-driven simulations (maximum buffer size of 50; the trace was collected from a *ns-2* simulation run that used a maximum buffer size of 50 and a bottleneck link speed of 1.5 Mbps), where we fixed the bottleneck link capacity (i.e., link A) at $1.5, 1.8, 2.2, 2.6$ and 3 Mbps, respectively ("$\diamond$"); (ii) another set set of 5 open loop trace-driven simulations



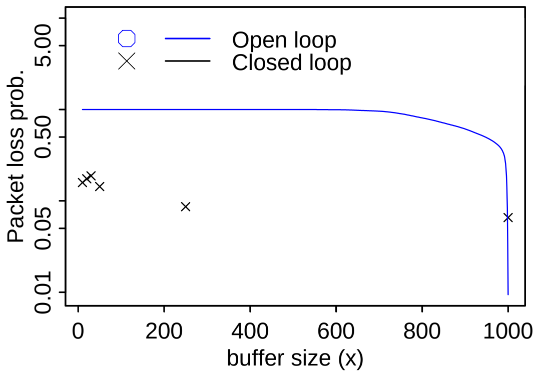


Figure 7: Open loop vs. closed loop, infinite sources (left) vs. Web sources (right): Loss probability vs. buffer size.

(maximum buffer size of 50; the trace was collected from a *ns-2* simulation run that used again a maximum buffer size of 50 but had a bottleneck link speed of 3 Mbps), where we consider the same set of a bottleneck link capacities ("$+$"); and (iii) a set of 5 closed loop (i.e., *ns-2*) simulations, all with a maximum buffer size of 50, but with different bottleneck link capacities, namely $1.5, 1.8, 2.2, 2.6$ and 3 Mbps, respectively ("$\times$"). Note that while the case (i) simulation results significantly underestimate the actual losses (i.e., the case (iii) results) and are generally way too optimistic, the case (ii) simulations significantly overestimate the actual losses and gives rise to overly conservative performance predictions. Both of these effects are due to the TCP feedback flow control mechanism that is explicitly ignored in the open loop trace-driven simulations. To illustrate, if TCP notices that there exists available bandwidth, it will in general allow the individual connections to increase their sending rates. On the other hand, if TCP receives too many indications of network congestion (via dropped packets), it has mechanisms in place that ensure that the offered load will be reduced. Similar results (not shown here) hold for performance measures other than loss probability, or when varying, for example, the maximum buffer size instead of the bottleneck link capacity.

Finally, we comment on another commonly-used approach for inferring buffer dynamics in an Internet-like setting from open loop trace-driven simulations, where in addition, the workload model comes into play. It is common engineering practice to use the the complementary probability distribution of the buffer occupancy in an infinite buffer queue as an accurate substitute for the loss probability in a finite buffer system. To check the validity of this practice in our simple networking setting, we run a number of identical *ns-2* simulations, except that we considered different maximum buffer occupancies at node 3, namely $10, 20, 30, 50, 250$ and 1000, and obtained the actual loss probabilities as a function of the buffer size at the node 3 queue. As our infinite buffer system, we take the simulation with maximum buffer size of 1000 and infer from it the complementary probability distribution function that the buffer exceeds a certain value x. The results are depicted in Figure 7 (the solid lines correspond to the infinite buffer approximation, while the crosses indicate the actual loss probabilities), where the left plot is for the infinite source case and the plot to the right is based on the Web sources. For the infinite sources and a maximum buffer size of 1000, the queue length process resulting from the *ns-2* simulation run (i.e., closed loop) turns out to tightly fluctuate around 725. In fact, the maximum buffer capacity is never exceeded, resulting in no losses and implying that in this scenario, the sources are bandwidth limited on their access links. In contrast, the *ns-2* simulation for the 1000 buffer size case with Web sources does occasionally fill the whole buffer resulting in significant losses of more than 5%. On the other hand, note that the infinite buffer approximations are by definition open loop based and predict, for example, almost 100% packet losses for the 50 buffer case, even though the actual packet losses are below 15% for Web sources and below 3% for the infinite sources. Overall, Figure 7 illustrates that the infinite buffer approximation can lead to extremely conservative performance prediction, making this open loop-based approach to inferring aspects of a closed-loop system essentially useless. The same conclusion holds true if we consider other specifications of our networking configuration and can be backed up by a careful study of the connections' TCP states in the different scenarios (details of such a study will be discussed in a forthcoming paper).

5 Conclusion

Even though the networking configuration considered in our simulation experiments is admittedly unrealistic and oversimplified, based on our findings, a word of caution is in place when generalizing results obtained in environments with infinite sources and open loop systems to real networks such as the Internet, where a major part of the traffic is generated by Web users and uses TCP. In fact, we have illustrated with a few examples that such generalizations can lead to highly conservative if not meaningless performance predictions. While infinite source models and open loop toy examples can provide deep insight into and physical understanding of the dynamics of real networks, we believe that their credibility could be substantially enhanced by demonstrating that the insight and understanding they provide (i) remain essentially unchanged, or (ii) may require appropriate modifications, or (iii) are no longer applicable when accounting for realistic variability at the source level and when taking the impact of closed loop feedback controls into serious consideration.

Looking ahead, it will be interesting to see whether or not some of the generic differences observed in our abstract setting will remain valid for more realistic network configurations. Another less obvious shortcoming of our experiments presented in this paper is that we completely ignore the potential of feedback from the network all the way back to the application layer; that is, the congestion state of the network may have a direct impact on our Web-user source model because it may directly influence the Web-browsing behavior of individual users. While there exists mainly anecdotal evidence for the presence of such types of feedback behavior (e.g., Internet "storms" [8]), we have seen little empirical evidence for the widespread existence of such feedback in our analysis of a wide variety of Internet traffic measurements. Nevertheless, the potential pitfalls associated with assuming an open loop characterization at the source level should be kept in mind and may require revamping the current approach to source modeling, depending on how the Internet develops in the near future. Other aspects not considered in our experimental studies concern replacing the drop-tail queueing discipline in the router by, say, RED, for "random early drop" [7]; incorporating TCP features such as SACK (selected ack) or delayed acks [10]; and dealing with the problem of two-way or cross traffic (e.g., see [17, 4]). Part of our ongoing efforts to understand the dynamics of TCP traffic in a realistic networking setting deals with some of these aspects and how they impact our current understanding, and will be published elsewhere.

Acknowledgments

We would like to acknowledge Polly Huang for her help with the *ns-2*-simulations.

References

[1] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, S. Shenker, K. Varadhan, H. Yu, Y. Xu, and D. Zappala. Virtual InterNetwork Testbed: Status and research agenda. Technical Report 98-678, University of Southern California, July 1998.

[2] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance'98/ACM Sigmetrics'98*, pages 151–160, 1998.

[3] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic - evidence and possible causes. In *Proceedings of ACM Sigmetrics'96*, pages 160–169, 1996.

[4] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proc. of the ACM/SIGCOMM'99*, pages 301–313, Boston, MA, 1999.

[5] A. Feldmann, A. C. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic. In *Proc. of the ACM/SIGCOMM'98*, pages 25–38, Vancouver, B.C., 1998.

[6] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The changing nature of network traffic: Scaling phenomena. *Computer Communication Review*, 28(2):5–29, 1998.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congstion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[8] B. A. Huberman and R. M. Lukose. Social dilemmas and Internet congestion. *Science*, 277:535–537, 1997.

[9] V. Jacobson. Congestion Avoidance and Control. In *Proc. of the ACM/SIGCOMM'88*, pages 314–329, 1988.

[10] V. Jacobson, R. Braden, and D. Berman. Tcp extensions for high performance, May 1992. Request for Comments 1323.

[11] V. Paxson and S. Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.

[12] S. Shenker, L. Zhang, and D. Clark. Some Observations on the Dynamics of a Congestion Control Algorithm. *ACM Computer Communication Review*, 20(4):30–39, October, 1990.

[13] W.R. Stevens. *TCP/IP Illustrated Volume 1*. Addison-Wesley, 1994.

[14] W. Willinger, V. Paxson, and M. S. Taqqu. *A Practical Guide to Heavy Tails: Statistical Techniques for Analyzing Heavy Tailed Distributions*, chapter Self-similarity and heavy tails: Structural modeling of network Traffic. Birkhauser Verlag, Boston, 1998.

[15] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5:71–86, 1997.

[16] L. Zhang and D. Clark. Oscillating Behavior of Network Traffic: A Case Study Simulation. *Internetworking: Research and Experience*, 1(2):101–112, 1990.

[17] L. Zhang, S. Shenker, and D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proc. of the ACM/SIGCOMM'91*, pages 133–147, 1991.