Experiences with Network Simulation

Lawrence S. Brakmo and Larry L. Peterson

Department of Computer Science
University of Arizona
Tucson, AZ 85721-0077
{brakmo,llp}@cs.arizona.edu

Abstract

Simulation is a critical tool in developing, testing, and evaluating network protocols and architectures. This paper describes x-Sim, a network simulator based on the x-kernel, that is able to fully simulate the topologies and traffic patterns of large scale networks. It also illustrates the capabilities and usefulness of the simulator with case studies. Finally, based on our experiences using x-Sim, we identify a set of principles (guidelines) for network simulation, and present concrete examples that quantify the value of these principles, along with the cost of ignoring them.

1 Introduction

As the Internet becomes an integral part of our daily lives, major changes in its infrastructure are being proposed. These changes are designed to support the increasing traffic and to provide the services necessary for future applications. It is imperative that we have tools to analyze and compare these proposed mechanisms before they are deployed. Since the Internet is a large, production network, it is not practical to conduct controlled experiments directly on the network itself. This means we need simulation environments powerful enough to model the Internet's behavior, as well as analysis tools to help us understand the results.

Most large networks exhibit very complex behavior as a result of three basic factors: (1) subtle protocol interactions, (2) complicated network topologies, and (3) complex traffic patterns. The last two factors have an obvious impact on simulation—we must be able to simulate topologies with hundreds of hosts, and the simulator must be able to recreate the traffic patterns found on real networks. The first factor is, in many respects, the tricky one. This is because individual protocols behave in unexpected ways when combined with other protocols to implement a complete network. To make matters worse, variations in the implementation of a single protocol—even when it conforms with the abstract specification of the protocol—can have a dramatic impact on the protocol's behavior and performance. Finally, protocol behavior and traffic patterns are highly inter-related. In short, the simulator must facilitate working with actual protocols, rather than just abstract specifications of protocols.

This paper describes a suite of tools to simulate and analyze network and protocol behavior. The centerpiece of this suite is x-Sim, a network simulator based on the x-kernel [6] which satisfies the requirements enumerated in the previous paragraph. Section 2 describes the simulator—as well as the related analysis tools—and compares it to existing simulators and emulators. Section 3 then gives two case studies that illustrate the usefulness of the tools in doing performance debugging. Next, Section 4 describes some very practical lessons we have learned about doing realistic network simulations. These lessons serve to re-enforce general rules of simulation, highlighting their relevance to network simulation. Finally, Section 5 offers some conclusions.

2 Simulator and Analysis Tools

When designing a network simulator, one must balance the issues of accuracy and execution time. In general, higher accuracy results in slower simulations. However, as computing power continues to increase, the cost of the simulations becomes less significant, but never irrelevant. Our approach is to create a simulator that is as accurate as we can make it, and then to introduce optional mechanisms to increase the speed (lower the accuracy) as the need arises. This approach, of going from more accurate to less accurate simulations, allows us to gauge the effect that the lower accuracy has on the simulations. Then we can decide if the resulting lower accuracy simulations are good enough to be useful.

SIGMETRICS '96 5/96 PA, USA
© 1996 ACM 0-89791-793-6/96/0005...$3.50

This section first describes the structure of x-Sim in terms of three fundamental components—*links*, *nodes*, and *load*. It then compares x-Sim to existing simulators and emulators, presenting the relative strengths and weaknesses of each approach. Finally, it discusses the analysis tools.

2.1 Links

Physical links can usually be simulated in software very accurately, at least to the level of the written specification. Simulating a link's real behavior accurately may entail some analysis of real implementations to find, among other things, the likelihood of losses due to noise. For example, accurately simulating an ethernet should include the ability to simulate collisions and exponential backoff. As another example, simulating ATM networks implies the use of small packets, increasing the complexity level—and execution time—of a simulation.

The architecture of x-Sim closely models the interactions between a link and the host/network adapter. The lowest software layer in each node (driver) is tightly coupled to the type of link (network) to which it is connected; together they model the interactions that occur in a real computer between the network, the adaptor, and the device driver.

The three main operations of the driver/network interace in x-Sim are: (1) query the network, for example to check if it is busy; (2) send a packet, and (3) callback the host to model interrupts. For example, a node connected to an Ethernet would first query the network to see if it is busy. If it is not, it can then send the packet. Otherwise, it registers a callback to be notified when the network is not busy. Once the packet has been transmitted successfully, the driver gets a callback notifying it of this fact so it can delete its copy of that packet. However, since Ethernets allow collisions, the interactions can become more complicated. For example, a query may return that the network is not busy when it occurs right after another host has started transmitting, resulting in a collision of packets. In this case, both hosts get a callback at some later time indicating that the transmission failed. Each host then schedules a retransmission at a later time based on an exponential backoff mechanism.

The simulator currently supports point-to-point links, Ethernet links (with and without collisions), and generalized multiple-access links. The first and last type of links allow the user to specify several parameters, including the bandwidth, delay, byte overhead per packet, a minimum and maximum packet size, and a loss or corruption probability. The second link type exactly models an Ethernet.

2.2 Nodes

Physical links are used to connect network nodes, which are usually either routers or hosts. To capture their behavior, we need to consider the characteristics of both the processor and the software, thereby simulating both the delays introduced by the nodes, and the algorithms that they run.

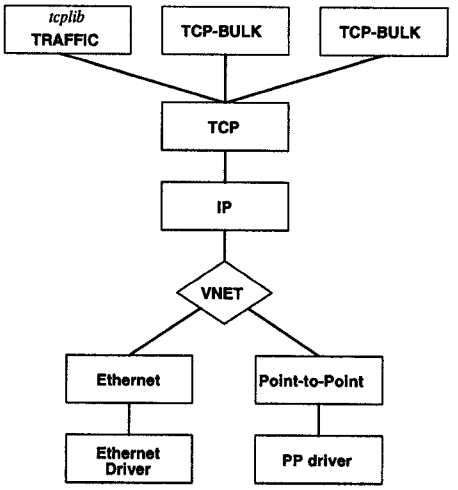

Figure 1: Example Protocol Graph.

In x-Sim, the software running on a node is represented in one of two ways: (1) by an x-kernel protocol graph (an example is shown in Figure 1), or (2) a program that implements an abstract specification of the node. When using a protocol graph (case 1), each protocol in the graph is given by the actual C code that implements the complete protocol. In other words, the former case can be viewed as supporting a *direct execution* simulation. Thus, a router can be simulated with an actual implementation of IP, or by an abstract specification that models some queuing discipline, such as FIFO. Note that when a protocol is given as a module in the protocol graph, then this is exactly the same implementation that runs in the x-kernel. This makes it possible to move a protocol between a simulated network and a real network.

To simulate the processor accurately, one needs to account for the delays resulting from the execution of the code invoked to handle each packet. For an even more accurate simulation, one needs to also add some jitter to these delays, which is be the result of interactions between different subsystems. Simulating the delays and jitter with total accuracy is probably not practical, as the jitter may be due to cache and TLB effects. Even if one could simulate all of these details, it may not be the best thing to do as it ties the results of the simulation to a specific model of hardware. x-Sim addresses this issue by allowing the protocol code to specify delays during which the node does no other processing; incoming packets are not be processed during such a delay. These delays might be based on measurements of the code running on an actual processor and can be specified by a uniform or normal distribution.

2.3 Load

Network load is usually the result of many applications executing on numerous hosts. In the case where one is interested in studying the behavior under realistic load, realistic traffic sources are of the utmost importance. For example, traffic

created by multiple large TCP transfers is radically different from traffic generated by multiple FTP, TELNET, SMTP and NNTP connections.

The first step for having realistic traffic is to use a common implementation of the underlying protocols—x-Sim gets this from the protocol graph. Next, we need a way to model the traffic. Here, x-Sim contains a protocol for simulating Internet traffic based on *tcplib* [4]. *Tcplib* is a library that models Internet traffic sources based on empirical data collected at different Internet gateways, and it has been shown to produce realistic traffic patterns [13]. This traffic model is realized as a protocol called TRAFFIC, which sits at the top of the protocol graph; i.e., this protocol generates work for TCP.

The TRAFFIC protocol starts conversations with interarrival times given by an exponential distribution. Each conversation can be of type TELNET, FTP, NNTP or SMTP and expects a set or parameters. For example, FTP conversations are parametrized by the number of items to transmit, control segment sizes, and the individual item sizes. All of these parameters are obtained from *tcplib* and are based on probability distributions obtained from real traffic traces. Each of these conversations runs on top of its own TCP connection, meaning that each conversation adapts to the network conditions according to TCP's congestion control algorithm.

One last feature of the traffic simulation protocol is that it is instrumented in such a way that the major traffic characteristics are independent of the underlying protocols and router congestion mechanisms—the traffic itself is of course affected by these differences. What we mean by major traffic characteristics are the start times of each conversation, the type of conversation, and the parameters for that conversation. This feature allows us to compare the effects of changes—e.g., to the TCP implementation—have on the traffic. Thus, we can determine the effects on the routers and link utilization when we go from the one implementation of TCP to another.

Finally, there are other protocols (everything is a protocol in the x-kernel) that model other types of transfers; for example, TCP bulk transfers.

2.4 Related Work

Some of best known freely available network simulators are REAL [10], Netsim [5] and the recently released ns from LBL's Network Research Group[1]. These simulators do not support common implementations of protocols (direct execution). Instead they contain code that simulates the major characteristics of the protocol. One problem with this approach is that it misses some of the behavior present in the common protocol implementations. In particular, it is our experience that in evaluating the different BSD implementations of TCP, running the actual TCP code is preferred to running an abstract specification of the protocol; the latter is mostly useful for rapid experimentation.

For example, we have found the actual behavior of timers in the BSD implementations of TCP to be quite important. Specifically, the round-trip time (RTT) measurements are done with a clock that ticks every half a second, resulting in very coarse estimates of the RTT. This affects the real-time behavior of the protocol because retransmit timeouts tend to be much longer than the actual RTT. Many existing simulators use more accurate RTT measurements, resulting in retransmission timeouts that are much smaller than those in BSD TCP. The consequence is that packet losses that result in timeouts have a lesser effect on the throughput in the simulator than it does in the BSD implementation.

As another example, both the Netsim and REAL implementations of TCP adhere to only the basic congestion control algorithm: the window increases linearly and decreases multiplicatively, and RTT measurement and timeouts are based on fine-grained clocks rather than the 500ms granularity present in the BSD implementations of TCP. More importantly, it would be very hard, if not impossible, to do a full implementation of TCP under either framework due to architectural constraints. As a result, neither simulator could be used to study the exact behavior of one of the common implementations of TCP (like the BSD implementations) to see if there are any problems with it.

Another important feature is that since x-Sim protocols are just x-kernel protocols, one can use the simulator while implementing and debugging a new protocol, and when done, just move it to any system based on the x-kernel infrastructure to be tested under real network conditions.

An alternative to simulation is emulation [1], a technique that uses workstations connected by a real network, with modifications to the operating system to simulate slower links and larger propagation delays. Under this framework, simulated time and real time are the same. Some of the advantages of emulators are that they run real protocols, actual behavior like processing overheads need not be simulated, and a two minute simulation only takes two minutes to run. There are some disadvantages, however, such as the cost of the necessary hardware (to emulate a network with five hosts you need at least five workstations), the fact that you cannot emulate a host or a link that is *faster* than the actual hardware, and the real-time behavior of the software is tied to specific hardware. A final disadvantage has to do with the available timer resolution, which can result in artifacts affecting the accuracy of the emulation. For example, an emulator with timer accuracy of 1ms implies that one cannot emulate the exact delay involved in sending packets smaller than 500 bytes on a 500KB/s link.

Finally, it is not really an issue of simulation versus emulation. Simulation, emulation, and tests on the actual network should all be considered part of the practitioner's repertoire. The confirmation of one technique's results with another technique serves an important function in the scientific process.

[1] See http://www-nrg.ee.lbl.gov/ns/.

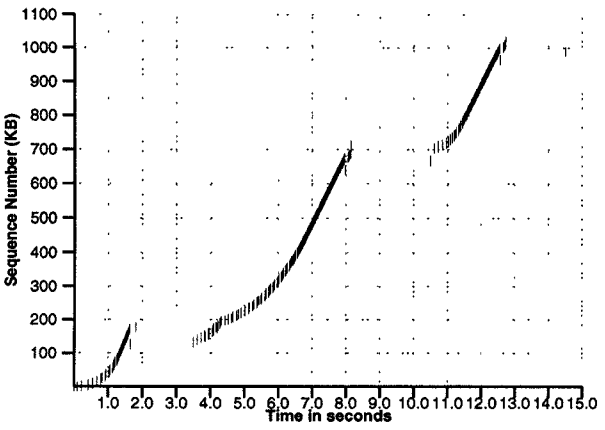

Figure 2: Time-sequence plot of TCP-Lite with no other traffic (throughput: 70KB/s).

2.5 Analysis Tools

Running realistic simulations is only the first step, and not necessarily the most time consuming step. Analyzing the results usually involves more work than running the simulations themselves, emphasizing the need for tools to aid in this analysis. For the rest of the section we describe some of the tools available with the x-Sim simulator.

These tools can be divided into two levels. At the high level, some of the protocols write summary information to a database at the end of each experiment. For example, TCP bulk transfers save the number of kilobytes successfully transmitted, the kilobytes resent, the number of TCP timeouts, the largest timeout, and so on. Routers save average link utilization, average queue sizes, number of packets and bytes lost.

At the low level, some protocols and routers can also save detailed trace information. These traces are analyzed and displayed using a set of tools that are part of the x-Sim package. The most detailed traces are obtained from the more complex protocols, and they have proven invaluable in our analysis and understanding of these protocols. For example, the x-kernel/x-Sim versions of the different TCP implementations have been augmented with calls to trace all the relevant state of the protocol, such as the size of the different windows and buffers, the time when packets are sent or received, information about retransmitted data, and so on.

These traces can be examined either graphically, showing the main characteristics of a protocol or router, or in full detail as a list of events. The common procedure is to first examine the traces graphically, then if anything interesting or unusual shows up, to examine the traces in full detail.

Examples of the graphical output from the tools are shown in the next section, where we consider some case studies showing the usefulness of the simulator.

3 Case Studies

This section gives two case studies that illustrate how we have used x-Sim and the associated analysis tools to performance debug TCP [15].

The most widely used implementations of TCP—Tahoe, Reno and Lite—are named after the BSD Unix distributions they were a part of. Each succeeding generation, starting with TCP Tahoe and ending with TCP Lite, was the result of adding more features to its predecessor. For example, TCP Reno augments TCP Tahoe with a better loss recovery mechanism, called Fast Retransmit [14], which resulted in throughput improvements of 10 to 20%. TCP Lite augments TCP Reno by implementing the *Big Window* and the *Protection Against Wrapped Sequence Numbers* options [8].

As will be seen later in this section, newer implementations of TCP not only include desirable features but also sometimes introduce performance bugs. Hence the need for tools with which to analyze the behavior of TCP connections. Graphical tools are generally the most useful.

Figure 2 shows a time-sequence graph of a particular TCP transfer. Each mark in the graph represents the time when a packet was sent. For example, the mark at coordinates (t, s) indicates that a packet whose payload starts with the s^{th} byte was sent at time t. Time-sequence plots have been the standard way of examining the behavior of TCP connections. From these plots we can see, among other things, when packets are retransmitted (as seen by vertical gaps in the sequence of marks), and when timeouts occur (as seen by large time intervals when no packets are sent).

As useful as time-sequence graphs are, it is our experience that a graph showing most of the internal state of the protocol is more informative. Such a graph could be used to analyze not only the coarse behavior of TCP, but also to study the details of a particular implementation. There is, of course, a price to be paid as a result of increasing the level of information—a steeper learning curve. After a long period of experimentation, we converged to the top graph shown in Figure 3.

TCP is a window based protocol. It uses windows to control the number of bytes that the sender is allowed to have *in transit*. A byte is considered *in transit* if it has been sent but has not yet been acknowledged by the receiver.

The top graph in the figure is a time plot of the value of these windows. For example, the light gray line shows the congestion window. This window is used for congestion control, and is increased either exponentially (when starting the connection or when restarting after a pause), or linearly (all other times). The thin black line, usually following the congestion window, indicates the exact number of bytes that are *in transit*. Its value must always be less than or equal to the other windows.

The top graph in Figure 3 shows a lot more information than just the value of the windows used by TCP. Due to a lack of space, we can only describe the most important features

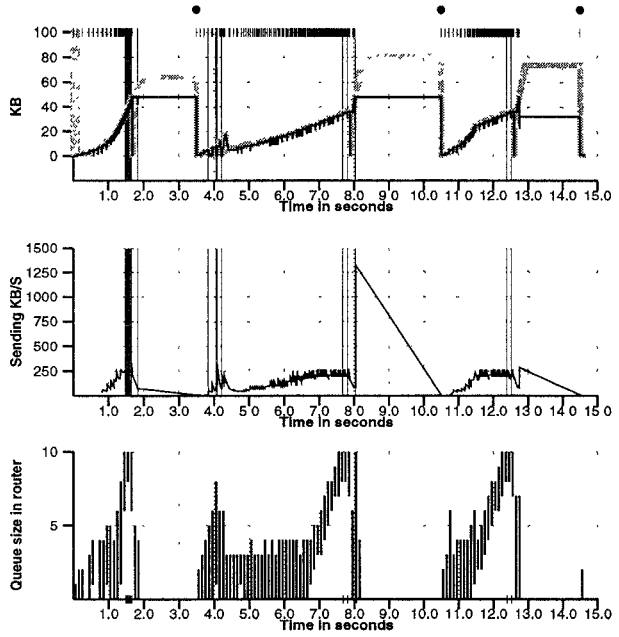

Figure 3: TCP-Lite with no other traffic (throughput: 70KB/s).

here. The circles at the top of the graph represent retransmit timeouts (RTOs), the thin vertical lines going from the top to the bottom of the graph show the time that a packet, which is later retransmitted, was originally sent. The small tic marks at the 100KB level indicate the time when packets were sent; they are analogous to the tic marks in Figure 2.

We are aware that anyone looking at this graph for the first time may be confused by its complexity. There are options to limit the amount of information that is displayed in these graphs; however, we wanted to show how much information can be displayed in a clear way, at least to a trained eye familiar with the workings of TCP.

3.1 Analysis of BSD-Lite TCP

As mentioned earlier, one of the strengths of the simulator is its ability to execute some of the most common protocol implementations. For example, the Tahoe, Reno and Lite implementations of TCP.

As the first step in the analysis, we created a simple network consisting of two ethernets joined by a point-to-point link supporting a 200KB/s bandwidth and 50ms delay. A host in one of the ethernets sends one MB of data to a host on the other ethernet. Even though there is no other traffic present, the average throughput is only 70KB/s.

The plots in Figures 2 and 3 were taken from the traces of this transfer. We have already described the meaning of the top plot in Figure 3. The middle plot shows the average sending rate in KB/s. Note the huge spike in the sending rate at 8 seconds, which results in a timeout at 10.5 seconds. Finally, the bottom graph shows the queue size at the router. Each vertical bar represents the range of the queue size during a 100ms interval. For example, we see that between 4.0 and 4.1 seconds the minimum queue size was one, and the maximum queue size was eight.

The features we have described are sufficient to uncover likely problem areas in the behavior of TCP-Lite. For example, we see that the timeout periods when no data is sent are lasting between 1.5 and 2.5 seconds. Given that the round trip times (RTT) are less than 200ms, it seems worthwhile to do a more detailed analysis of the retransmit timeout (RTO) mechanism in TCP-Lite.

Another sign of possible problems is the huge spike in the sending rate at eight seconds. Given that the bottleneck bandwidth is 200 KB/s, it seems unwise to send packets at a rate of 1250 KB/s; the cause for this burst of packets can be seen in the top graph (for a detailed explanation see [2]). Finally, the last indication of trouble is the fact that there are so many RTOs. This implementation of TCP contains the Fast Retransmit and Fast Recovery mechanisms; these mechanisms are supposed to prevent the type of timeouts we see in the time interval between 12 and 15 seconds.

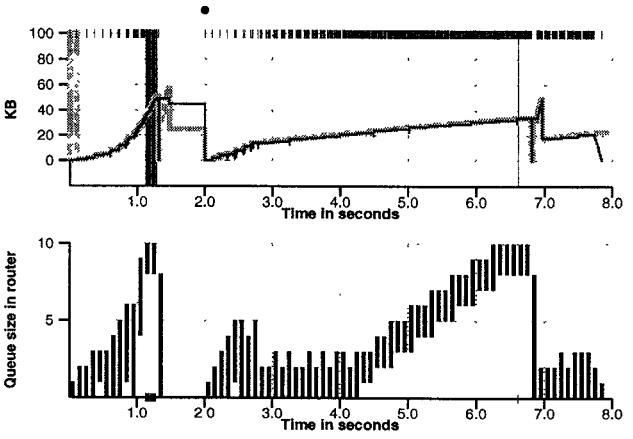

Figure 4: TCP-Lite with fixes and no other traffic (throughput: 128KB/s).

The analysis of the detailed traces uncovered a series of problem with the BSD-Lite implementation of TCP. Some of the problems, like the cause of the sending rate spike, were caused by bugs in the code (specifically a bug in the header prediction code). Other problems, like the duration of the RTOs, were caused by suboptimal algorithms. All together, more than five problems were uncovered [2]. Figure 4 shows the behavior, under the same scenario, of a modified implementation of TCP that had these problems fixed. This is the plot that someone familiar with both TCP and our graphs would have expected to see initially.

To gauge the full effect of the modifications we also ran more complex simulations consisting of more complex topologies where some of the hosts were sources of *tcplib* based traffic. These simulations showed that one could expect a 20% increase in throughput, as compared to the original

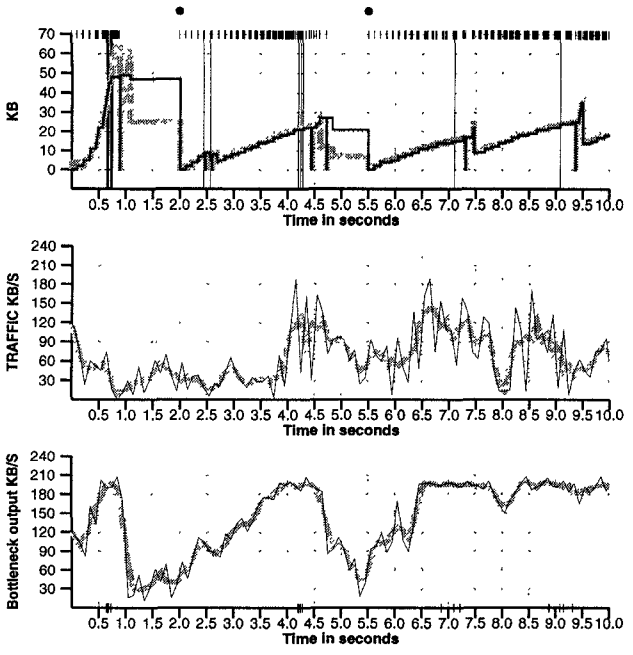

Figure 5: TCP bulk transfer sharing a link with *tcplib* generated traffic.

implementation, in wide area connections.

3.2 Analysis of TCP Congestion Control

TCP's standard approach to full bandwidth utilization is to continually increase the amount of bytes *in transit* until losses occur. At this time, TCP responds to the packet loss by reducing the amount of bytes *in transit* by half. The only sign of congestion TCP is able to interpret is packet loss. Therefore, the interplay of both mechanisms, full bandwidth utilization and congestion control, result in a periodic pattern of congestion and recovery periods. That is, TCP must *create* congestion to recognize that there is no more bandwidth available, even when it is the only source of traffic.

We can see this behavior in Figure 4. In the time interval between 3 and 6.5 seconds, we see an increase in the amount of data *in transit* from 16KB to 38 KB. At the beginning, from 3 to 4.2 seconds, we see an increase from 140 to 200KB/s in the sending rate,[2] as a result of the increase of the amount of data *in transit*. Since the available bandwidth is only 200KB/s, any further increase in the amount of data *in transit* has to be absorbed by the bottleneck buffers, as seen in the bottom graph.

Figure 5 shows the behavior of TCP when it is sharing a link with *tcplib* based traffic. The middle graph shows the traffic level, the bottom graph shows the bottleneck link utilization; the link's maximum bandwidth is 200KB/s. It is easy to see, for example in the period between 5.5 and 7 seconds, that TCP continually increases its congestion window, and thus the amount of data *in transit*, regardless of what the background traffic is doing. TCP is oblivious of other traffic until the network gets so congested that packets are lost.

It certainly would be preferable to have a congestion avoidance mechanism that detects the incipient stages of congestion, so it can avoid congestion rather than control it. The earlier observation that an increase in the amount of data *in transit* cannot result in an increase of the throughput beyond the available bandwidth is a useful notion tha can form the basis of a congestion avoidance mechanism.

The general idea is as follows, we first define a *predicted* bandwidth as a function of the amount of data *in transit* and we compare it to the *actual* bandwidth (sending rate). The difference between these two bandwidths is proportional to the amount of router buffer space consumed this connection. All that remains is to set buffer utilization goals that are then used to trigger either an increase or a decrease of the congestion window.

This congestion avoidance mechanism was incorporated into a new implementation of TCP, named Vegas, with the goal of testing its effectiveness at preventing congestion [3]. The simulator proved invaluable as a debugging, tuning, and measuring tool. The fact that protocols can be moved between the simulator and platforms supporting the x-kernel allowed us to easily measure the performance of TCP Vegas on the Internet. Both sets of measurements, in the simulator and on the Internet, showed a major decrease in losses and a large increase in throughput—TCP Vegas had more new mechanisms than just congestion avoidance.

4 Guidelines for Network Simulations

There is a wealth of literature on the subject of simulation; for example, see [9] and [11] for a good overview. Among the issues presented in the literature are general principles all simulation experiments should follow to insure the validity of the results. Typically, these principles are presented in general terms, and it is up to the experimenter to apply them to their particular case.

This section goes beyond these general principles to present some very concrete guidelines dealing with *network* simulations, in particular. Most of these guidelines involve the application of a well known rule, but they are worth enumerating because they contain realistic evidence of what can happen when these rules are not followed. That is, these examples help to quantify the value of the general simulation principles, as well as the cost of ignoring them. It is our observation that many papers in the networking literature ignore some of these guidelines.

Most of the examples used in this section were obtained from the two simulation networks shown in Figures 6 and 7. The first, shown in Figure 6, consists of a simple topology with 7 nodes. The second simulated network, shown in

[2] Since TCP is a self-clocking protocol—i.e. it uses the reception of acknowledgments to trigger new transmissions—its sending rate is closely related to its throughput.

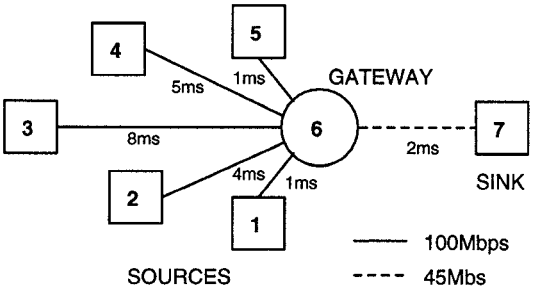

Figure 6: Simple Simulation Network.

Figure 7, is much more complex.[3] It consists of 16 traffic sources, each consisting of two or three hosts running *tcplib* based traffic. The rectangular boxes represent sources of "bulk data" transfers. The resulting traffic consists of nearly a thousand new connections being established per simulated second, where each connection is either a TELNET, FTP, SMTP or NNTP conversation.

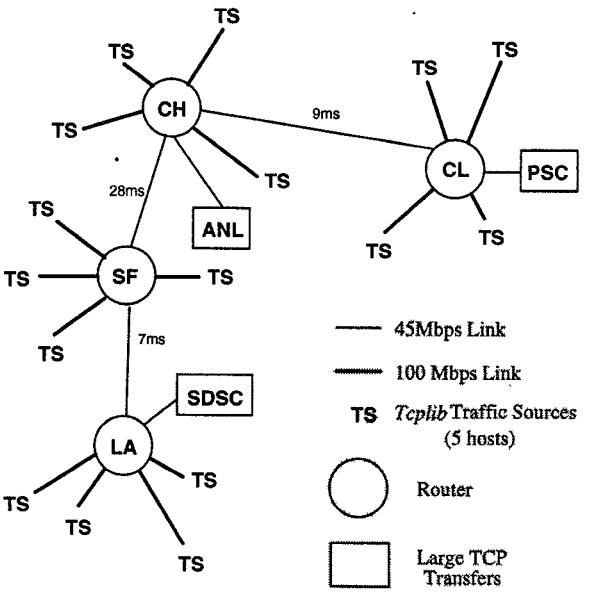

Figure 7: Complex Simulation Network.

4.1 Sensitivity to Network Parameters

Protocols and communication networks generally deal with bits and packets, both of which are discrete entities. Many protocol components, such as retransmit timers, also behave in a discrete manner. Many network parameters, such as the number of buffers at a router, are discrete quantities. Due to this discrete nature, small changes in the parameters defining a network experiment can have a considerable effect in the outcome of an experiment. The following two examples illustrate the sensitivity of a simulation to different settings of such discrete values.

The first example involves the number of buffers at a router. Here, we are running an experiment that simply measures the throughput TCP can achieve across a simulated network operating under some load. In the BSD implementation of TCP, time is measured by counting the number of times a timer fires. As data is being sent over the connection, losses due to congestion result in a temporary halt in the transmission. In some cases, if there are multiple losses, the transmission is restarted only after a retransmit timeout. This delay generally lasts for two ticks of the 500ms clock. There are situations were for one number of buffers at the router, the transmission hiatus (gap) starts right *before* the timer fires, and since the hiatus only lasts for two ticks of the clock, the result is the hiatus lasts for about 500ms. Increasing the number of buffers by just *one* can result in delaying the start of the transmission hiatus until *after* the timer fires, so now two clock ticks represent about 1000ms rather than 500ms.[4]

For example, we have seen simulations where increasing the number of buffers from 22 to 23 leads to a 34% decrease in throughput.

A second example of the unexpected effects that changes in the simulation parameters can have is seen when running two TCP transfers at the same time. In this case, the experiment consists of two competing TCP transfers, with some delay between the start of each transfer. For a particular simulation, a delay of 170ms between the start of the two transfers resulted in a throughput of 76.1KB/s for the first transfer and 76.2KB/s for the second transfers. Changing the delay to 150ms results in throughputs of 87.8KB/s and 75.9KB/s, respectively, a change of more than 15%.

These examples demonstrate the sensitivity of a simulation to seemingly trivial differences in the settings of certain network parameters. The guideline is to vary the simulation parameters over a range as large as possible. The basic principles behind it are: (1) simulate the right cases, (2) perform sensitivity analysis, and (3) give confidence intervals with your results.

4.2 Analyzing Results

One should also look at the results from every experiment to catch any instances in which the results vary greatly from the average. These instances may expose problems in the experiment, problems in the algorithms, or problems with the implementation that surface only sporadically. Looking just at averages may result in missing vital information. In our experiments, we depend heavily on graphs that allow us to quickly see the important results from all the experiments, not just the averages.

Graphs of the average queue size (number of buffers

[3] This network models a subset of the ANS backbone. It has major routers at Los Angeles, San Francisco, Chicago and Cleveland. Three supercomputer centers provide sources of large "bulk data" transfers.

[4] The same effect can also be obtained when keeping the number of buffers constant, but by changing the time when the test starts relative to the tick of the 500ms clock.

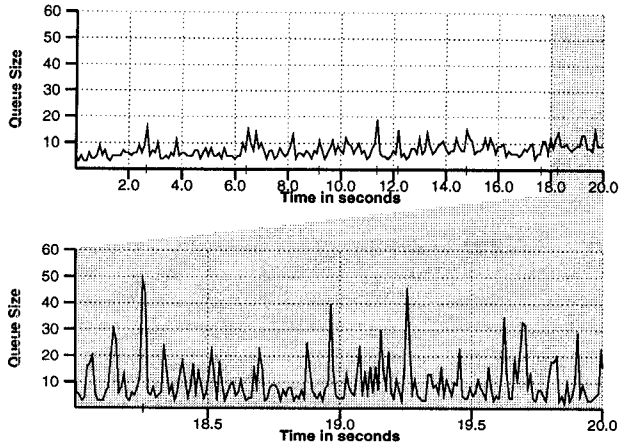

Figure 8: Graph of queue size at bottleneck showing information lost due to averaging.

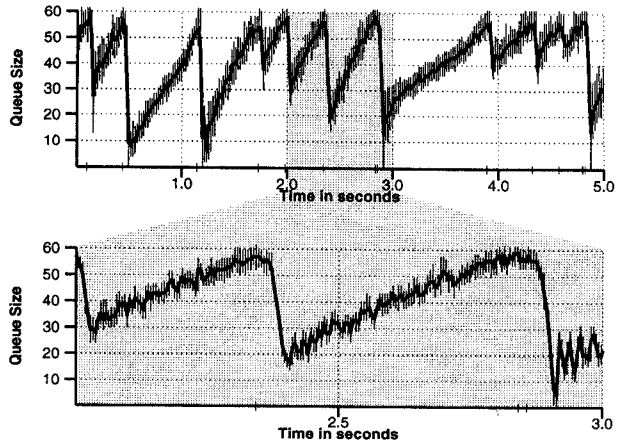

Figure 10: Queue behavior during four large TCP transfers.

in use) at a bottleneck router is our first example of where looking just at the averages can be misleading. One of the important uses of a simulator is to experiment, observing the effects on the experiment of modifying the parameters. This type of experimentation helps develop intuition that can be used to both find problems and solutions. If we are interested in congestion control mechanisms at the router, it is very important to have a clear understanding of queue behavior under different traffic patterns. The top graph in Figure 8 shows the average queue size during a 20 second period of a simulation, where each point represents the average queue length over a 100ms interval. Note that under the conditions of our simulation, there are between three and ten thousand packets arriving at the bottleneck router every second, so it is not practical to show the graph of the instantaneous queue size.

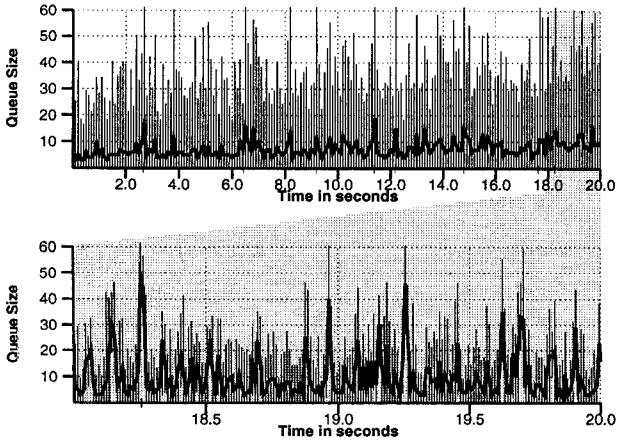

Figure 9: Graph of queue size at bottleneck including min-max bars.

This graph indicates that the queue size doesn't vary much, and that it stays under 20 buffers. However, if we look at the bottom graph, which shows the time interval between 18 to 20 seconds but averages over a 10ms interval (rather than the 100ms interval), we see that the queue size reaches all the way to 50 and that it goes over 20 quite often. At this level of detail, we can see a markedly different behavior of the queue. Obviously, one should show both the maximum and minimum queue size (over the average interval) together with the average. Figure 9 shows the same graphs as Figure 8 but with the added information. Each vertical bar represents the queue size range over the average interval (100ms for the top graph, 10ms for the bottom graph). The top graph now tells us a whole new story—even though the average queue size is small, it varies dramatically during each interval. The mechanism of showing the range together with the average allows us to get a much clearer picture and keeps us from jumping to the wrong conclusions.

The basic principle is to look at distributions and limits, not just means, when analyzing results.

4.3 Realistic Traffic Sources

Traffic sources are an integral part of most network simulations. The most common ways to simulate this traffic has been either through Poisson sources, or by having multiple "bulk transfer" TCP connections.

There are several problems with Poisson sources. First, it has been shown that Poisson sources fail to accurately model either LAN traffic [12] or WAN traffic [13]. Moreover, Poisson sources are non-reactive—they behave the same way regardless of the network behavior. For example, unlike TCP sources, which modify their sending rate under the presence of losses, Poisson sources don't modify their rate or any aspect of their behavior for that matter when packets are dropped due to congestion. Another example of non-reactive traffic sources are playback sources, which create traffic based on traces collected from real systems.

There are two problems with these approaches. The first one, which applies only to Poisson sources and not to play-

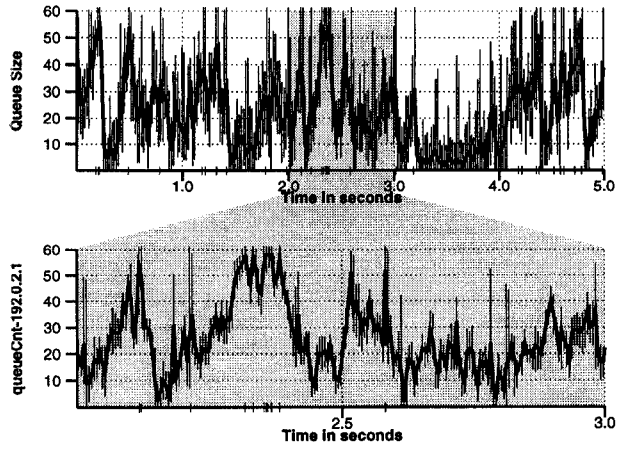

Figure 11: Queue behavior during two large TCP transfers and *tcplib* traffic.

back sources, is that the simulated traffic is a poor model of real traffic. The second problem, which applies to both Poisson and playback sources, is a result of their non-reactive behavior: there is no way to measure the effect the new connections have on the background traffic. This is an important issue, as new protocols should not be adopted before their effect on current traffic is known.

The other common way of simulating traffic—starting multiple bulk transfers—does produce reactive traffic, but the dynamics of this type of traffic are very different from those of real traffic. For example, Figure 10 shows the queue size at the router of the network in Figure 6 when nodes 1-4 are bulk transfer TCP sources. The graph shows a simple pattern and no information is gained by changing scales in the bottom graph. Compare this graph to the graph in Figure 11, in which hosts 1, 3 and 4 are *tcplib* traffic sources, and hosts 2 and 5 are bulk transfer TCP sources. The behavior at the router is totally different, and changing scales shows new detail.

The degree of dissimilarity between the two graphs should serve as a warning against using bulk transfers as a traffic source when simulating wide-area networks. Moreover, studies of congestion control mechanisms should include traffic sources that are more realistic than either Poisson or bulk transfers.

A final issue regarding realistic traffic sources is the need for traffic reproducibility. This means that a traffic source should always produce the same traffic pattern when given the same set of initial parameters, regardless of the underlying protocols, the number of nodes, or the type of network. By traffic pattern we do not mean that a packet containing x bytes is transmitted at time t. Instead, we refer to the structural characteristics of the traffic. For example, when dealing with *tcplib* based traffic, a traffic pattern would be specified by the time a conversation starts, the type of the conversation (FTP, TELNET, SMTP and NNTP), and the parameters for that type of conversation. Traffic sources that meet this definition of reproducibility allow us to compare things such as the effect on the traffic's throughput or losses when it is running on top of two different protocols.

The basic principles behind these discussions are to always use realistic traffic sources and justify any distributions used in the experiments.

4.4 Insider Knowledge

When doing experiments involving TCP transfers, it has been a common practice to set TCP's send buffer size based on the bandwidth delay product of the connection. After all, why use more buffering than is really needed? The size of the send buffer limits the maximal throughput a connection can achieve for a given RTT. The result of having this upper limit below the available bandwidth is obvious: the achieved throughput is not necessarily a reflection of the protocol or the network, but is an artificially imposed limit.

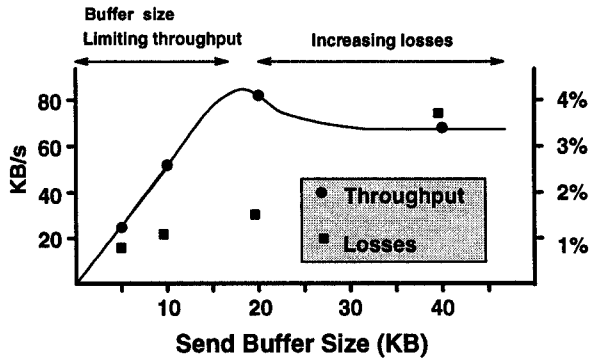

Figure 12: Throughput vs. Send Buffer Size.

As the send buffer size increases, there is a corresponding increase in the throughput achieved by the TCP transfer. This relationship continues until a little after the size of the send buffer equals the average available bandwidth-delay product—using a few buffers at the bottleneck router allows the connection to take advantage of transient increases in the available bandwidth. What happens after that point depends on the TCP implementation. In the BSD implementations of TCP (i.e. TCP Reno), the throughput starts to decrease, and does so until it reaches a point where additional increases of the send buffer size have no effect on throughput (see Figure 12).

The decrease in the throughput as the buffer size increases is the result of the bandwidth probing mechanism in the BSD-based implementations of TCP. These implementations increase the window size continuously, by about one packet per RTT, until the buffer size limit is reached or packets are lost. As the window size increases from its optimal point, which uses only a few buffers at the bottleneck, the TCP connection uses more and more buffers. The more buffers the connection uses, the higher the likelihood that the router will not be able to absorb a packet burst. Since some of

these bursts will be by the connection itself, the larger number of buffers used by a connection increases the likelihood that the connection will have losses. When the losses are detected, the congestion control mechanism in TCP reduces the window size by half, reducing the throughput.

Figure 12 also shows the average throughput achieved by TCP Reno as a function of the send buffer size during a series of transfers over the Internet between two fixed hosts.[5] Notice how the losses increase as the send buffer size increases. Thus, using a send buffer size that limits the throughput would lead to misleading comparisons between protocols, as they would both perform equally under this condition. Similarly, using the optimal size would also lead to misleading comparisons, unless this fact is pointed out, as real applications will not be using this optimal send buffer size—it is difficult to know this size since different connections have different optimal sizes, and this optimal size changes with time.

This is another example where varying the parameters of the experiment leads to useful insights and comparisons. For example, a set of experiments comparing TCP Reno with TCP Vegas [3] showed that TCP Vegas was not sensitive to the send buffer size, as long as it is big enough so that it is not the limiting factor.

The basic principle to remember is that simulation results are constrained by the values of the input parameters. Any bias in the choice of these parameters, such as insider knowledge, will affect the results.

4.5 Byte versus Time Transfers

Throughput is generally measured by sending a specific amount of data and recording the time it took to send it. If we are interested in measuring the difference in throughput between two protocols, one of which is generally faster than the other, then there is the possibility of erroneous measurements in certain cases.

When simulating traffic, there may be a systematic tendency for the level of traffic to either increase or decrease during the duration of the transfer. If the level increases during the transfer, then the slower protocol will end up being measured as being much slower than it really is because it had to compete with higher levels of traffic at the end of the test. Alternatively, if the level of the background traffic tends to decrease during the duration of the test, then the slower protocol will be measured as being faster—relative to the faster protocol—than it really is.

Another way of measuring the throughput is to instead transfer as much data as possible for a certain amount of time. This way both protocols will encounter the same background traffic for the duration of the transfer. As we found early on, the differences in the measured throughput between the two methods can be considerable, depending on the characteristics of the traffic. For example, in one series of experiments we measured a 56% difference in the throughput between two protocols when sending a full megabyte of data, but only a 28% difference when sending as much data as we could in 30 seconds. Clearly, the preferred method should be to use timed transfers rather than length transfers.

In summary, throughput is a function of the number of bytes sent in a certain period of time. When comparing the throughput of two different mechanisms, you can either measure how much time it takes each to send the same amount of data, or how much data each can send in a certain amount of time. In most cases, the latter is preferred.

4.6 Uniformity

It is very easy for simulations to loose a lot of the variability (randomness) that is inherent in the real world. This can be either a design flaw with the simulator or the result of badly chosen simulation parameters. An example of uniformity in a simulation is starting all transfers at the same time. This is very easy to do in a simulator, but it would generally be an uncommon occurrence outside of a simulator.

Another example of a lack of variability can occur with protocols, like the BSD implementation of TCP, where coarse timers are used to perform special processing. In this implementation of TCP, two coarse timers are used, one firing every 200ms and the other every 500ms. Most of the implementations of TCP delay ACKs, which means that when they receive a packet they do not send an acknowledgment immediately, but instead wait for more packets to arrive so they can acknowledge more than one packet at a time. The BSD implementations of TCP waits for one more packet, thereby acknowledging two packets at a time. However, since another packet may not arrive, TCP uses the 200ms timer to send any acknowledgments that were delayed. The 500ms timer is used to check to see if packets need to be retransmitted.[6]

If one is not careful in the implementation, it is easy to have these timers fire at the same time on all hosts in the simulated network. This results in periodic bursts of data as all hosts send the delayed acknowledgments or start retransmitting lost packets at the same time. Since the retransmissions occurring as a result of the 500ms timers involve the slow-start mechanism, which is a restart of the connection involving exponential growth of the sending rate, the effect is equivalent to starting multiple transfers at the same time.

A similar burstiness can occur even when the timers for each node do not all fire at the same time. The problem happens when using realistic traffic sources involving multiple TCP connections, but having only a few traffic sources, each of which is responsible for a large percent of the traffic. Then, when one of the timers fires at one of the sources,

[5] Note that in the case where one does a lot of WAN transfers between the same two hosts, it may be worthwhile to find a send buffer size that results in near-optimal throughputs. This size should be re-evaluated every so often, as it is a function of the traffic.

[6] The exact mechanism for deciding when to retransmit is outside the scope of this paper; [14] gives an excellent description.

a large number of connections may send delayed ACKs or restart their transmissions.

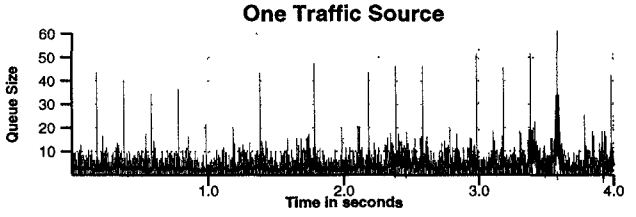

Figure 13: Graph showing artificial periodic effects.

A tool that is very useful at finding unwanted uniform behavior is Fourier analysis. This type of analysis uncovers periodic behavior that may point to possible problems and to the cause of these problems. Figure 13 shows the queue size at the router in Figure 6 when the only source of *tcplib* traffic is node 4. It is clear from the graph that periodic behavior occurs five times per second (every 200ms). The fact that the spikes occur every 200ms points to the fast timer in TCP as the cause of these spikes. Note that this behavior is not evident if we only look at the average queue size rather than the maximum queue size.

One way to eliminate these bursts is to have more traffic sources, so the bursts will be distributed more evenly. This assumes, of course, the traffic sources do not fire their timers at the same time.

The basic principle is to put a distribution on everything that needs it. The problem is that when dealing with complex simulators, it is not always clear what things need to have distributions in them.

5 Conclusions and Future Work

We have described x-Sim, a network simulator whose goal is to accurately model the different elements present in real computer networks. It is our experience that the ability of the simulator to execute real protocol code will be of assistance in testing and debugging new network components.

We have also presented a set of concrete guidelines one should consider when doing network simulations, together with examples illustrating the effects of ignoring these guidelines. These examples serve to quantify the cost of certain simplifications, such as simpler traffic sources.

We are currently working on improvements to the simulator to handle multicasting. This goal will be achieved by applying the MBONE modifications to the simulator's IP protocol. We are also working on the addition of full ATM support into the simulator. Finally, we are planning to add the necessary infrastructure to the simulator to support direct execution of BSD protocol code, in addition to x-kernel protocols.

References

[1] J.-S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Experience with TCP Vegas: Emulation and Experiment. In *Proceedings of the SIGCOMM '95 Symposium*, Aug. 1995. In press.

[2] L. S. Brakmo and L. L. Peterson. Performance Problems in BSD4.4 TCP. *ACM Computer Communication Review*, 25(5):69–86, Oct. 1995.

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

[4] P. Danzig and S. Jamin. tcplib: A Library of TCP Internetwork Traffic Characteristics. Technical Report CS-SYS-91-495, Computer Science Department, USC, 1991.

[5] A. Heybey. The network simulator. Technical report, MIT, Sept. 1990.

[6] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan. 1991.

[7] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM '88 Symposium*, pages 314–32, Aug. 1988.

[8] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. Request for Comments 1323, May 1992.

[9] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for ExperimentalDesign, Measurement, Simulation and Modeling*. John Wiley and Sons, Inc., New York, 1991.

[10] S. Keshav. REAL: A Network Simulator. Technical Report 88/472, Department of Computer Science, UC Berkeley, 1988.

[11] A. M. Law and W. D. Kelton. *Simulation Modelling and Analysis*. McGraw-Hill, New York, 1990.

[12] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic. In *Proceedings of the SIGCOMM '93 Conference*, pages 183–193, Oct. 1993.

[13] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. In *Proceedings of the SIGCOMM '94 Conference*, pages 257–268, Aug. 1994.

[14] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Publishing Co., New York, 1994.

[15] USC. Transmission control protocol. Request for Comments 793, USC Information Sciences Institute, Marina del Ray, Calif., Sept. 1981.