On the Effective Evaluation of TCP *

Mark Allman
NASA GRC/BBN Technologies
21000 Brookpark Rd. MS 54-2
Cleveland, OH 44135
mallman@grc.nasa.gov

Aaron Falk
PanAmSat Corporation
One Pickwick Plaza
Greenwich, CT 06830
afalk@panamsat.com

Abstract

Understanding the performance of the Internet's Transmission Control Protocol (TCP) is important because it is the dominant protocol used in the Internet today. Various testing methods exist to evaluate TCP performance, however all have pitfalls that need to be understood prior to obtaining useful results. Simulating TCP is difficult because of the wide range of variables, environments, and implementations available. Testing TCP modifications in the global Internet may not be the answer either: testing new protocols on real networks endangers other people's traffic and, if not done correctly, may also yield inaccurate or misleading results. In order for TCP research to be independently evaluated in the Internet research community there is a set of questions that researchers should try to answer. This paper attempts to list some of those questions and make recommendations as to how TCP testing can be structured to provide useful answers.

1 Introduction

[PF97] gives a thorough overview of the tradeoffs of different ways to examine internetwork performance and various considerations that should be taken into account when running internetworking experiments. This paper extends the thoughts outlined in [PF97] to the specifics of performing research on the Transmission Control Protocol (TCP) [Pos81]. Understanding the performance of TCP is important because it is the dominant transport protocol used on today's Internet. TCP employs a set of end-to-end congestion control algorithms [Jac88], which are the focus of a number of research papers and presentations[1]. There are numerous methods for evaluating TCP's performance in various environments and the impact of modifications to TCP's congestion control algorithms. The manner in which an experimenter studies TCP has a direct impact on the relevance and usefulness of the results obtained and the conclusions derived.

When initially explained TCP appears to be a fairly simple and easy to understand protocol. Furthermore, it would seem fairly straightforward to make some small change to the algorithms and assess the costs and benefits of this change. However, there are many subtleties to TCP's mechanisms, dynamics, and interactions with other traffic sharing a network path. In our experience with running TCP experiments we are constantly surprised by the interactions and dynamics that various TCP experiments illustrate. Thorough exploration into the behavior of TCP, the behavior of proposed changes to TCP and the impact of these changes on competing network traffic is needed before a research study will be taken seriously and have an impact on changing the protocol. As we review papers and listen to presentations we are often dismayed at the types of TCP experiments being performed and the conclusions derived from the results. This paper is not meant to discourage TCP research. Rather, we hope to encourage researchers to carefully think about their TCP experiments such that the investigations are solid and will have a positive impact on the continuing evolution of TCP.

Finally, we are also often dismayed by the lack of details in reports and presentations about TCP experiments. As with all scientific endeavors, details are important for thorough review and analysis of TCP proposals[2]. Without such details it may be difficult for researchers to understand the results of TCP experiments and therefore the tests become less compelling.

2 Choosing a TCP I: The Forest

The first question that researchers need to carefully consider before embarking on a TCP experiment is: *Which TCP should I use?* Often this question implicitly breaks into two (or more!) questions when carefully considered. This section considers which TCP should be used as a "baseline" against which experimental TCPs are compared, and which TCP should be used as an experimental TCP.

For the remainder of this section we will distinguish between today's current TCP implementations, denoted TCP_c, and the expected future TCP implementations, denoted TCP_f. While no two TCP implementations are identical, most are similar in the algorithms implemented, so consider TCP_c to be a current, widely-used TCP implementation that contains a typical set of TCP algorithms. While section 3 will outline the specifics of the current TCP_f, for this section just assume that TCP_f includes a number of widely-agreed upon and standardized TCP enhancements that will likely be implemented in the near future in most major TCP implementations. The difference between TCP_c and TCP_f is the time it takes for standardized TCP improvements to gain wide-spread deployment. Note that generally a number of experimental TCP_f implementations are available for researchers to use before such implementations become wide-spread in commercial operating systems. Next, we use TCP_e to denote a TCP implementation containing a researcher's experimental modi-

*This paper appears in ACM Computer Communication Review, October 1999.

[1] In this paper, we use the term "TCP" to include both the details of the TCP protocol itself and also the congestion control algorithms used by TCP.

[2] A nice reminder about the importance of details and the scientific method can be found in [Fey85].

fication. Finally, we use TCP_o to denote a TCP containing other enhancements, possibly competing, that have been proposed in the literature and may or may not be adopted in future versions of TCP. Often it is useful to compare the impact of TCP_e with various versions of TCP_o, especially if the TCP_e and TCP_o are attempting to solve similar problems.

Often the first decision a researcher must make is which TCP to use as a "baseline" against which a TCP_e will be compared. It is tempting to simply pick an "out of the box" TCP_c to obtain a baseline. However, such a choice can produce misleading results for two reasons. First, if the TCP_c chosen is overwhelmingly used as a client in the Internet, it may show less than optimal performance when pressed into service as a server. Likewise, a TCP_c that is used mainly on servers may exhibit poor performance when used as a client. The researcher could cope with this problem by using two TCP_c implementations, a popular client implementation and a popular server implementation. The second (more fundamental) reason why a TCP_c does not make a good baseline is that it likely does not include any of the performance enhancements that are expected to be widely deployed in TCP_f implementations. Since the enhancements included in a TCP_f are currently standardized and in the implementation process, a new TCP modification is not likely to be deployed before TCP_f and therefore comparisons to TCP_c are clearly not as compelling as comparisons to TCP_f implementations.

Note that a comparison of a TCP_e stack to a TCP_c implementation is not entirely without merit. Such a comparison can give the community insight into the performance improvements we can expect in the future, as compared to the current installed base. However, such a comparison *does not generally provide a compelling reason for changing TCP*.

The second decision a researcher must make is which TCP implementation to change and use as TCP_e. Given the above argument that building on a TCP_f is best, the researchers choices are often somewhat limited in that, as defined, TCP_f is not widely deployed. In addition, note that many times the operating system choice is further limited to only those operating systems that release source code. While the TCP enhancements in TCP_f may not be completely tested, we suggest that researchers look for TCPs that have been proven to be stable over some period of time in production networks. This will cut down on the number of bugs in the implementation and increase the strength of the results produced. In all cases, the researcher should carefully examine the TCP stack chosen, as discussed in section 5.

3 Choosing a TCP II: The Trees

This section details the items researchers should currently look for in a TCP implementation. Note that this list is subject to change with time and that a good understanding of the current state of the protocol is needed to answer the question of which TCP should be used. The current state of many popular TCP implementations is given by [Mah99].

The following items are standards track IETF mechanisms that we recommend be in all TCP implementations used in research experiments.

- **Basic Congestion Control.** We strongly suggest the TCP chosen for a given piece of research should contain the slow start, congestion avoidance, fast retransmit and fast recovery congestion control algorithms, as outlined in RFC 2581 [APS99]. Slow start and congestion avoidance are required by the IETF standards, while fast retransmit and fast recovery are recommended, mainly as performance enhancements. These algorithms provide TCP with standard end-to-end congestion control (as originally outlined in [Jac88]) and are widely implemented in most versions of TCP.

- **Extensions for High Performance.** The size of the socket buffers that should be used in TCP experiments is discussed in detail in section 6. Here we note that the standard TCP header [Pos81] limits the advertised window size (generally derived from the receiver socket buffer size) to 64 KB, which is not adequate in many situations. Equation 1 defines the minimum window size (W bytes) required for a TCP to fully utilize the given amount of available bandwidth, B bytes/second, over a network with a round-trip time (RTT) of R seconds [Pos81].

$$W = B \cdot R \tag{1}$$

  Therefore, a network path that exhibits a long delay and/or a large bandwidth may require a window size of more than 64 KB. RFC 1323 [JBB92] defines the *window scaling* extensions to TCP that allow the use a window size of more than 64 KB. Window scaling can lead to more rapid use of the TCP sequence space. Therefore, along with window scaling the Protect Against Wrapped Sequence Numbers (PAWS) algorithm is required. In turn, the PAWS algorithm requires the timestamp option (also defined in RFC 1323). The timestamp option adds 12 bytes to each segment. These additional header bytes are expected to be costly only to excessively low bandwidth channels. The timestamp option also allows TCP to easily take multiple RTT samples per round-trip time. As outlined in [JBB92], this is beneficial in obtaining an accurate estimate of the RTT, which is used for determining when to retransmit a given segment.

  The high performance options are currently being implemented in many popular operating systems and are expected to be in wide spread use relatively soon. While some experiments may not require these extensions a researcher needs to carefully examine the scenario under investigation and utilize these extensions when necessary.

- **Selective Acknowledgments.** As originally defined, TCP exchanges only gross information about which segments have been successfully received and which have not. TCP uses a cumulative acknowledgment (ACK) that simply indicates the last in-order segment that has arrived. When a segment arrives out-of-order a *duplicate ACK* (i.e., an ACK covering the last in-order piece of data received) is transmitted. A cumulative ACK provides no information about which segment triggered it. Therefore, the TCP sender does not have enough information to intelligently figure out exactly which segments have arrived.

  The selective acknowledgment (SACK) option [MMFR96] allows the TCP receiver to inform the TCP sender of which segments have arrived and which segments have not. This allows the TCP sender to intelligently retransmit only those segments that have been lost. In addition, it decouples the determination of which segment to transmit from the decision about when it is safe to resend a packet (from the perspective

of congestion control). This decoupling is discussed in detail in [FF96, MM96b].

The SACK option discussed in [MMFR96] only specifies the format for exchanging selective acknowledgment information. There are several SACK-based loss recovery algorithms that use this information to intelligently augment TCP's traditional loss recovery techniques. A conservative fast recovery replacement is outlined in [FF96]. Additionally, the forward acknowledgment (FACK) [MM96b, MM96c] and rate-halving [MSML99] algorithms offer alternative SACK-based approaches to replace fast recovery. RFC 2581 [APS99] allows the use of all the above algorithms.

The SACK option is currently being implemented in many popular operating systems and is expected to be in widespread use soon. Therefore, we recommend that researchers always include a TCP implementation with a SACK-based loss recovery algorithm.

- **Delayed Acknowledgments.** RFC 1122 [Bra89] allows that a TCP can refrain from sending an acknowledgment for each incoming data segment, but rather should transmit an ACK for every second full-sized data segment received. If a second data segment is not received within a given timeout (not to exceed 0.5 seconds) an ACK is transmitted. This mechanism is widely deployed in real TCP implementations and therefore we recommend that it be used in future TCP experiments. Excluding this mechanism does not necessarily yield useless data and may be quite useful as a comparison to the delayed ACK case. However, turning off delayed ACKs without carefully considering the costs and the impact on the conclusions drawn should be avoided.

- **Nagle Algorithm.** The Nagle algorithm [Nag84] is used to combine many small bits of data produced by applications into larger TCP segments. The Nagle algorithm has been shown to reduce the number of segments transmitted into the network, but also interferes with the HTTP [BLFN96, FGM+97] and NNTP [KL86] protocols, as well as the delayed acknowledgment strategy [Hei97, MSMV99], thus reducing performance. The Nagle algorithm is contained in many current TCP implementations and should be used in TCP tests whenever possible.

The following mechanisms are *not* standardized extensions to TCP. However, they are currently receiving attention in the research community and including these mechanisms as part of TCP tests will provide valuable data towards to goal of understanding the interactions between these mechanisms and the currently standardized algorithms. In addition, such tests may be useful is deciding whether to standardize the following mechanisms.

- **Larger Initial Windows.** RFC 2581 [APS99] allows TCP to utilize an initial congestion window of 1 or 2 segments. RFC 2414 [AFP98] outlines an experimental TCP mechanism that would increase the initial congestion window to 3–4 segments, based on the segment size. While this mechanism is not standard, we encourage researchers to experiment with the change and compare it with the standard initial congestion window in the hopes that such experimental evidence will lead to a concrete decision on whether using a larger initial window is worthwhile.

- **Explicit Congestion Notification.** As outlined in [Jac88, APS99], TCP interprets segment loss as indicating network congestion. However, RFC 2481 [RF99] defines a method in which a router can send a TCP an explicit message stating that the network is becoming congested, rather than dropping a segment. This Explicit Congestion Notification (ECN) is further discussed in [Flo94]. RFC 2481 is currently an experimental mechanism within the IETF pending further study into additional dynamics introduced by this new method of signaling congestion. Researchers are encouraged to compare ECN capable TCP connections with non-ECN connections in their research.

4 Simulation, Emulation, or Live Internet Tests?

Another decision a researcher must make when considering how to evaluate TCP is whether to simulate a network in software entirely, to use a small testbed of hosts handling live packets, to use some hybrid of simulation and a testbed (live emulation), or to make transfers over the Internet. Each of these methods of testing TCP has benefits and can yield useful results. However, each method also has disadvantages and one may be more useful than another for a particular experiment. While there is no perfect way to test TCP, researchers should understand the tradeoffs between the testing methods and choose the most appropriate method for the given experiment. The researcher should carefully consider and note any expected consequences of choosing one method over the others when reporting results.

Also note that, generally speaking, a single set of experiments using one of these testing methods is not enough evidence to get a particular mechanism widely adopted. Rather, a variety of experiments using multiple test methods generally provides a more compelling argument in favor of a change to TCP. The following subsections discuss the advantages and disadvantages of each method of testing.

4.1 Simulation

Many successful TCP performance evaluations have been conducted via simulation (e.g., [FF96, Hoe96]). A large variety of simulators for modeling internetworking protocols exist, and are currently used by researchers. Some of these tools are OpNet [Tec], *x*-sim [BP96], the Network Simulator [Hey90], REAL [Kes88], *ns* [MF95], as well as specialized simulators written by various researchers for their own use. Picking a simulator to use is sometimes complicated. We encourage researchers to investigate various simulators and choose the one best geared towards the research at hand. For instance, one simulator may have better models of a certain type of channel than another simulator. Also, the TCP model included in the simulator should be considered when choosing a particular simulator (see sections 2 and 3 for a discussion of what to look for in a TCP implementation). Finally, note that simulation can only be used to concretely to say that a particular TCP change is badly broken and that a particular proposal is promising. That is, do not expect the networking community to accept TCP changes only on the strength of a simulation study.

The following is a list of the advantages of using a simulator to evaluate TCP's performance.

- Simulators are not equipment intensive, as only a single basic workstation is needed to run the simulations and

analyze the data.

- Simulators allow a researcher to easily examine a wide range of scenarios in a relatively short amount of time. As outlined in [PF97], examining a large range of scenarios is important for drawing general conclusions about the performance of a protocol like TCP.
- Simulation also provides a means of testing TCP performance across "rare" networks that a researcher does not have good access to use (e.g., a cross-country gigabit network).
- Simulators are not hindered by the physical speed of a given network and therefore can be used to investigate how TCP may perform in the faster networks in the future.
- Complex topologies can be easily created via simulation, whereas such topologies would not be easy to replicate in a testbed environment.
- Simulators give the researcher access to data about all the traffic transmitted in the simulation. This allows analysis into the impact a particular change to TCP has on competing traffic sharing the network.
- Simulators give the researcher an easy way to test the impact of changes to the routing and queueing disciplines on the performance of TCP. This can also be done in a testbed or emulation environment (as outlined in the next two subsections), however changing routers for live Internet tests is generally not possible.
- Simulators can give a researcher access to a specialized network before such a network is ever built. For instance, by using simulation, a researcher can assess the performance of TCP over a low-Earth orbit satellite constellation before the first satellite is launched.

While the advantages of using simulation to assess TCP's performance are many, there are several downsides to this method of testing, as follows.

- Some simulators use an abstract TCP implementation, rather than using code found in real operating systems (e.g., the one-way TCP implementation included in *ns*). Such code can still be used to effectively analyze TCP performance. However, the researcher should remember that this code is one step removed from the real world and therefore the results may not directly match up with similar tests conducted over a real network. In addition, most TCP implementations contain bugs [Pax97a, All97a, PAD+99], which may not be present in the simulation. Of course, the reverse is also true. Regardless of whether the TCP implementation is a real in-kernel implementation, or an abstract version of the TCP algorithms, researchers should carefully examine the TCP used in their experiments. This point is explored further in section 5.

  Some simulators use code that is based on a real implementation of TCP (e.g., *x*-sim). Researchers should take care to make sure that all fixes that have been identified and applied to the real implementation are also included in the version of the code present in the simulator (see section 5 for a larger discussion of this topic). Whether or not having "real" networking code in a simulator is debatable. There is a possible tradeoff between having a common implementation ("warts and all") and having a correct implementation (possibly less warts, but no real world experience behind it either).
- Simulators do not generally model non-network events that may impact TCP performance. For instance, an operating system's scheduler latency may have an impact on the burstiness of a TCP connection on a very fast network.
- Simulators sometimes make assumptions that shield the TCP experiments from effects that may occur in the real world. For instance, some simulators send packets of uniform size and therefore could not be used to investigate the interactions that have been found between the delayed acknowledgment strategy and the Nagle algorithm [Hei97, MSMV99]. While this is a shortcoming of using a given simulator, the key is for researchers to understand what a particular simulator does well and where it does not attempt to duplicate the "real world".
- Simulating competing traffic requires making many assumptions. Poor choices for competing traffic can drastically skew performance results. See section 9 for a discussion on generating competing traffic.

We suggest that anyone choosing to use simulation to assess TCP performance read [PF97] on some of the pitfalls of simulating the Internet and methods for coping with these pitfalls. In addition, [PF97] discusses several other considerations that are not discussed further in this paper (e.g., choosing a network topology).

4.2 Testing Real Implementations

While simulations can provide valuable insight into the performance of TCP, they are often not as illuminating as tests conducted with real TCP implementations over real networks. This section outlines several methods for testing real TCP implementations, including their costs and benefits.

When conducting experiments using real systems, as opposed to simulators, researchers should not focus on TCP to the exclusion of all other aspects of the system. For instance, if a TCP transfer is "slow," the cause should be investigated. TCP may be the culprit, but it is also possible that the application protocol (or implementation) or some component of the operating system may be to blame. For instance, some versions of HTTP introduce extra (idle) round-trip times into the transfer time. Using a different version of HTTP may help isolate this source of delay. As discussed in section 10, taking packet-level traces of the TCP transfers under consideration is the best way to ensure that the effects being investigated are indeed network related, rather than being a consequence of using a particular operating system or application.

Note that investigating operating system and application layer performance problems is also a worthwhile effort and should be pursued to the extent possible (this may vary with the availability of the source code). However, such investigations are beyond the scope of this paper.

4.2.1 Testbeds

Using real hosts on a small isolated testbed network can alleviate some of the problems with simulation and a number of effective TCP studies have been performed using testbeds

(e.g., [PBS+96]). In a testbed, real TCP implementations are being tested (bugs and all) over real networks. The experiments are therefore subject to things like scheduler delays and router packet processing time that are hard to model via simulation. In addition, testbeds can incorporate hard to simulate network components, such as a satellite link. On the other hand, testbeds are limited in their complexity and speed by the equipment on hand. Therefore, research utilizing testbeds can be much more expensive than research via simulation. Another large disadvantage of using a testbed is that if commercial operating systems are utilized the researcher generally has no way to modify the TCP code to test new mechanisms. The researcher can use one of the free Unix based operating systems (e.g., Linux). However, in doing so, the researcher may not be able to test the dominant TCP implementations used in the Internet.

4.2.2 Emulation

An *emulator* models a particular piece of the network path between two real hosts. Therefore, emulation is a mix between simulation and using a testbed. For instance, several researchers have turned a standard workstation into a router, passing packets between two physical networks, as if there was some network between the two real physical networks in the testbed [Riz97, Fal99]. This can allow testing of real TCP implementations over emulated satellite channels, for instance. A number of studies have utilized emulation to effectively evaluate TCP performance (e.g., [ADLY95, HK99]). Another advantage of using emulation is that the researcher can modify the queueing disciplines used in the routers and measure the impact these changes have on real TCP stacks, whereas researchers generally do not have access to alter the software running on commercial routers.

On the other hand, like simulation, emulation abstracts away some of the real behavior of the pieces of the network modeled. Also, real hosts running through an emulated network are still constrained by hardware speeds, as in the case of using a testbed for experiments. Finally, emulators may not be able to represent complex or changing topologies.

4.2.3 Live Internet Tests

One way researchers might choose to decrease the impact of an artificial environment, such as simulator or a testbed with an emulator, is to run experiments over the Internet. Such tests can be illuminating with regards to the behavior of TCP "in the wild," as opposed to in some network a researcher has created. There are several ways to conduct tests over the Internet.

The most complete method for testing TCP over the Internet is to use a mesh of N hosts that can each make TCP transfers to each other (yielding measurements over $O(N^2)$ network paths). This method of measuring the Internet is outlined in [Pax97a, Pax97b, Pax97c]. An architecture and the corresponding software for doing general purpose network measurements between a mesh of hosts is currently being developed by the "NIMI" project [PMAM98]. An additional consideration when choosing the N hosts for the mesh is that the hosts be linked to the Internet with a variety of types of connections (i.e., various bandwidth and various delay connections). If all N hosts in the mesh are similarly connected, the results obtained from the tests may not be valid for networks that are connected to the network in a drastically different way (e.g., testing TCP performance between Internet hosts connected via high-speed fiber provides no way to assess the performance of TCP over slow dialup modem links). Another consideration when choosing the hosts for the mesh is to choose different TCP implementations, such that the results are not biased by the particulars of any one TCP stack.

A downside of using NIMI-like environments is that often TCP researchers need to change the TCP implementation in the operating system. For instance, researchers may want to investigate a particular change to TCP's slow start algorithm that requires changing the TCP implementation in the kernel (e.g., [All98, All99]) . This may not be possible on a pre-existing mesh of hosts, as the kernel change may interrupt or invalidate experiments being conducted by other researchers. Also, setting up a mesh of hosts all running a custom-built kernel can be quite difficult and time-consuming. One way to mitigate this problem is to employ a user-level implementation of TCP, such as TReno [MM96a], rather than an in-kernel TCP implementation. However, this shares some of the disadvantages of modeling TCP faced by simulators (but, this time in a real network).

Another method researchers have used to study the Internet is to transmit traffic from a single host, H_0, on their local network to a large number of remote hosts, H_1–H_n, around the Internet. For example, a researcher might wish to capture all traffic to and from a local WWW server and analyze the performance of TCP using this data. While this method of testing TCP can provide some useful data, the data is not nearly as complete as when utilizing a mesh of NIMI-like hosts. On the other hand, data collected using this method is much easier to gather than data collected using a mesh of hosts. Using a single sender when testing TCP can bias the results based on the network behavior near the sender, H_0. For instance, consider the case when H_0's site is connected to the Internet via an X bits/second channel. The TCP transfers to the n Internet hosts cannot be made any faster than X bits/second, even if some of the n remote sites are better connected and could transfer data at a higher rate if H_0's link could support it. Finally, using a single sender can bias the measurements due to bugs or particular behaviors of the particular TCP stack chosen to transmit the data.

One of the disadvantages of conducting live experiments over the Internet is the inability to assess the impact the sending TCP has on the other network traffic sharing the network path. Whereas, with simulators and testbeds it is fairly easy to monitor all traffic on the given network, it is difficult to obtain the same kind of monitoring of all the traffic competing with the TCP transfer a researcher generates when running over the Internet. In addition, assessing the impact of a new queueing algorithm, or some other mechanism that is expected to be placed in the middle of the network is difficult to accomplish in tests conducted over the Internet.

5 Know Your TCP Implementation

One important aspect of running TCP experiments is to have a good understanding of the TCP implementation being used (whether an abstract implementation in a simulator, an in-kernel implementation or an implementation in a user-level diagnostic tool). As outlined above, researchers should understand which options and algorithms the TCP implementation in question contains and how those options relate to current state-of-the-art implementations. In addition, some widely used operating systems have well-known TCP bugs [Pax97a, PAD+99] that cause non-standard TCP behavior (e.g., a TCP that does not initialize congestion window prop-

erly may produce a large burst of data when a transfer begins, rather than using the slow start algorithm). These bugs should be fixed, if possible, as some of these bugs can have a large impact on performance. In general, researchers should take responsibility for validating that the TCP implementation chosen behaves as expected and not take the behavior for granted [Flo99].

If an abstract version of TCP is used, the researcher should attempt to understand the differences between the abstract implementation and real TCP implementations and what impact these differences may have on the results. For instance, the one-way TCP module included with the *ns* simulator uses a default clock granularity of 100 ms, while many real TCP implementations use clocks with 500 ms granularity. The behavior of *ns*, in this case, is not wrong according to the TCP standards. However, experiments conducted with *ns* (assuming the granularity is not changed from the default) may not highlight the performance implications of TCP timeouts to the same degree as experiments with an implementation that uses a coarser granularity clock.

Finally, the TCP congestion control algorithms, as outlined in [APS99] allow implementers some amount of latitude in implementing some of the small details. For instance, [APS99] specifically says that slow start should be used when the congestion window (*cwnd*) is less than the *slow start threshold* (*ssthresh*) and that congestion avoidance should be used when $cwnd > ssthresh$. However, the document does not specify which algorithm should be used when $cwnd = ssthresh$. These small details can have a subtle impact on measured TCP performance and should be fully specified such that others can recreate (if only in their mind) the experiments. A list of the under specified congestion control details is given in [MA99]. Note that depending on the TCP implementation used these details may not be available, but any that can be inferred should be reported.

6 Choosing a Window Size

As shown in equation 1, the maximum window size a TCP connection can utilize has a large impact on the resulting performance. The maximum window size is generally the minimum of the send socket buffer (set on the TCP sender) and the receive socket buffer (which generally determines the advertised window on the receiver). The TCP sender's *congestion window* (*cwnd*) may further limit the amount of data the sender can inject into the network, depending on the level of congestion in the network path.

If the maximum window size is too small relative to the available bandwidth of the network path, the TCP connection will not be able to fully utilize the available capacity. Alternatively, if the maximum window size is too large for the network path to handle, the congestion window will eventually grow to the point where TCP will overwhelm the network with too many segments, some of which will be discarded before reaching the destination. Therefore, it is tempting to calculate the window size needed via equation 1 and use that window size. This is often possible in the artificial environments in which TCP is tested. However, by setting the maximum window to the exactly appropriate value some of the naturally occurring behavior of TCP is lost. That is, the congestion window is increased to the right value and then never again changed. Also, in general it is very difficult to pick the appropriate window size for all network paths and levels of congestion. If the transfer is long enough, the TCP congestion control algorithms naturally find a congestion window size that is appropriate for the network path if the maximum window size is large enough to overwhelm the network. Therefore, artificially limiting the advertised window leads to less compelling results.

We suggest that researchers use automatic socket buffer tuning [SMM98] in their TCP performance evaluations. An operating system that uses *autotuned* socket buffers does not impose a single maximum window size on the connection, but rather the buffer sizes (and therefore, the maximum window size) grows with the congestion window. Therefore, the network path being used determines the maximum window size, rather than the maximum window size being an arbitrary limit placed on the connection by the end hosts. While autotuning is not widely deployed at the present time, it can be considered part of TCP_f and researchers can emulate the network behavior provided by autotuning by setting their send and receive socket buffers very large, such that the network dictates the behavior of TCP, rather than being limited by the end hosts.

One tempting comparison for researchers to make is to compare a very modern version of TCP, complete with their proposed tweaks, with an "out of the box" TCP with no changes to the default socket buffer sizes. As discussed in section 2, such a comparison may be interesting to show what TCP performance users may be able to expect in the future as compared to the performance experienced today, however such an experiment does not provide a compelling reason to implement the proposed change to TCP (without additional experimentation).

7 Choosing Data to Send

Another question that researchers must consider is how much data should be transmitted and what applications should be used to send the data. Transfers that are small compared to the delay-bandwidth product of the network have been shown to underutilize the available bandwidth (in [All97b], for example) and therefore experiments with short transfers only show a portion of TCP's behavior. On the other hand, very long transfers show TCP's steady-state behavior quite nicely, but the startup behavior is lost. While both short transfers and long transfers provide solid data points about TCP's performance, a study will have more impact if a variety of transfer sizes are taken into account. As recommended in [PF97], we suggest that researchers transfer a wide variety of file sizes to illustrate the performance of TCP (and any proposed changes) across an entire range of possible transfer lengths. We suggest using transfers ranging from several hundred bytes to hundreds of times larger than the delay-bandwidth product of the network path, with a generous number of points in between.

In addition to varying the size of the TCP transfers, researchers should carefully consider which application protocols should be studied. Simple FTP [PR85] transfers lend themselves to investigating TCP's behavior over a wide range of transfer sizes. However, HTTP [BLFN96, FGM+97] is currently the dominant application protocol on the Internet [TMW97] and has much different behavior than FTP. For instance, HTTP is likely to be used for small, interactive request/response transfers, whereas FTP is more likely to be used for bulk data transfer [TMW97].

Furthermore, simply choosing an application protocol is sometimes not enough. For instance, there are a number of different versions of HTTP defined and running on the Internet, and each of these versions has a slightly different sending pattern. Each of these patterns exercises the underlying TCP stack in a different way. For instance, some HTTP versions allow TCP connections to become "idle" waiting for additional

requests from the user. That is, the connection remains open between the two hosts, but no data is exchanged. Different TCP implementations differ in the way they start transmitting data after a long inactive period [VH97] and therefore the interaction between the HTTP implementation and the TCP implementation is important. So, even when choosing a particular application protocol, researchers should also take care in choosing and specifying an appropriate version of the protocol, (generally by what is prevalent in the Internet). Alternatively, any report should include a discussion about any performance differences that may be expected if a different version of the application protocol was used.

We suggest that researchers use FTP to measure the impact of a given proposal on a variety of transfer sizes. In addition, we recommend measuring the performance of HTTP traffic since WWW traffic is the most prevalent type of traffic on the Internet today [TMW97].

Note that successful TCP studies do not necessarily need to include transfers of each application type currently being used on the Internet to provide solid, usable results. However, researchers need to be clear when providing conclusions about their experiments that the conclusions are only for the applications tested (and similar application protocols) and could potentially be different if the application layer was changed.

Finally, note that as new application protocols and protocol versions are developed and start being used on the Internet, researchers should integrate these into their TCP performance tests. In other words, researchers need to keep their testing up-to-date with regards to the current applications that are being used on the Internet to provide strong, compelling TCP measurements.

8 Drawing Strong Conclusions from Single Flow Tests

Often times researchers attempt to draw strong conclusions about TCP performance by running a single flow over their testbed, emulated network or simulated network path. These tests can be very useful for deriving information about the theoretical behavior of TCP or a proposed change to TCP. In addition, using only a single flow makes diagnosing problems much easier than when attempting to diagnose the problems in a dynamic network with many competing traffic flows. However, such single flow tests are not particularly realistic and therefore, these tests *should not* be used to authoritatively say that a given proposal should be widely implemented. First, using a single connection on an otherwise unloaded network path does not take into account the impact of competing traffic on the TCP flow in question. Also, the impact of the TCP flow on the competing traffic cannot be assessed (the proposed change might make TCP unfair to other traffic, for instance).

Figures 1–3 show the behavior of an unchanged TCP (figure 1), denoted TCP_0, and two experimental changes to TCP (figures 2 and 3), denoted TCP_1 and TCP_2 respectively, in a wide variety of scenarios (various transfer sizes and various router queue lengths). These figures and a discussion of the proposed mechanisms can be found in [All98]. These figures are used in this paper as illustration, so the particular changes being proposed are irrelevant. The proposed changes shown are fairly minor and were not expected to radically change TCP's behavior. However, as shown in figure 2, the changes proposed in TCP_1 drastically alter the ideal TCP behavior in many circumstances. In this case, a set of single flow TCP tests yields a strong conclusion that the change in question is not beneficial to TCP performance. However, TCP_2 shows a fairly minor (yet, positive) change in TCP's behavior, as expected. While it may be straightforward to conclude that TCP_1 is not an improvement over TCP_0, this does not mean that a conclusion can be drawn that TCP_2 *is* an improvement over TCP_0, just that TCP_2 does not show the large drawbacks that TCP_1 shows. While figure 3 shows a nice result, these experiments cannot be used to definitively determine that the proposed change (TCP_2) should be widely implemented. This experiment does not answer many prerequisite questions to justify protocol changes such as: Does the change still represent a performance improvement when TCP_2 is sharing the network with other traffic? Does a TCP with this change compete fairly with a TCP without the change? Does the change pose any problems for other application protocols? Therefore, we can conclude that while TCP_2 is a promising change, the proposal needs further study before we can conclude that it is ready to be widely implemented and used in the Internet.

9 Introducing Competing Traffic

While tests involving only a single TCP flow in an otherwise unloaded network can yield interesting theoretical observations they cannot be used to illustrate the performance of TCP as it is used in real networks. Therefore, we recommend that researchers also test TCP performance in a more dynamic environment with competing traffic flows (both TCP and non-TCP traffic). Such tests can show more realistic effects, such as ACK compression [Mog92] or the effect of unpredictable loss patterns. In a dynamic environment we recommend researchers examine not only TCP's performance, but also the impact of the TCP modifications on the network in general (e.g., impact on competing flows, router queue sizes, etc.).

Choosing exactly how to model competing traffic flows is a difficult problem and researchers often use a simple, but inaccurate model. For instance, simply using a Poisson process to generate transfer sizes has been show to be inaccurate [PF95]. We strongly suggest that researchers read [PF97], which outlines several ways to cope with the difficulty of generating accurate traffic patterns, before conducting experiments with competing traffic.

10 Collecting and Analyzing Data

Many tools for evaluating various aspects of TCP performance have been written. RFC 2398 [PS98] discusses a number of popular TCP testing, analysis and visualization tools. RFC 2398 includes a brief discussion of each tool, including the tool's purpose, the systems on which the tool will run and where to obtain the program. The tools outlined in RFC 2398 represent a wide variety of TCP testing and analysis software (e.g., programs that coordinate TCP transfers, TCP data analysis utilities, tools for testing a TCP for conformance against the TCP specifications, etc.). Additionally, researchers can find various utilities for analyzing packet trace files from the Internet Traffic Archive [Arc]. We encourage researchers to be aware of the above tools and choose the appropriate tools for each set of experiments. Also note that a ready-made tool is not always available to perform the needed analysis for a given experiment and in that case the researcher should be prepared to craft a tool themselves.

The straightforward way to gauge the performance of TCP is to measure throughput, or the time required to transfer a given number of bits of data. While this is a very telling metric, it is not necessarily the only measurement that indicates

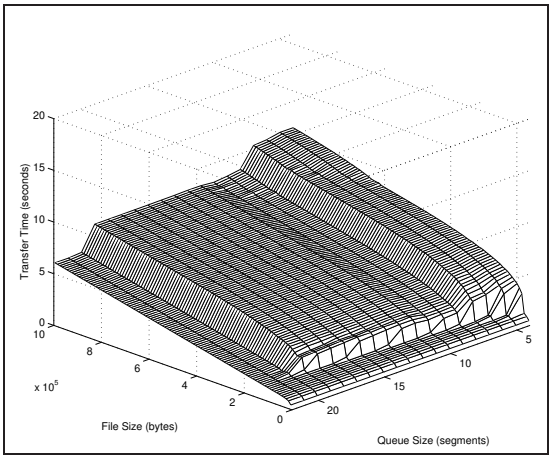

Figure 1: TCP_0 (standard TCP with no changes).

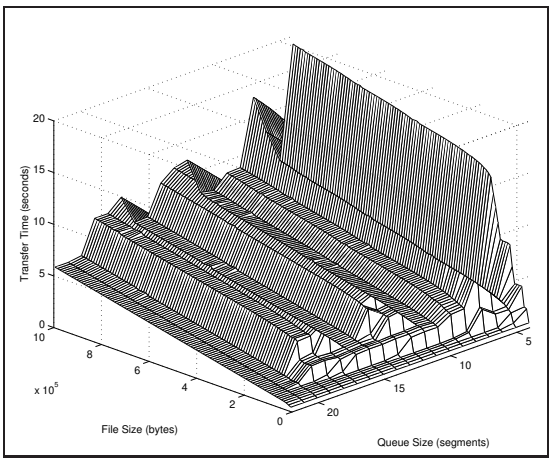

Figure 2: TCP_1 (first proposed change).

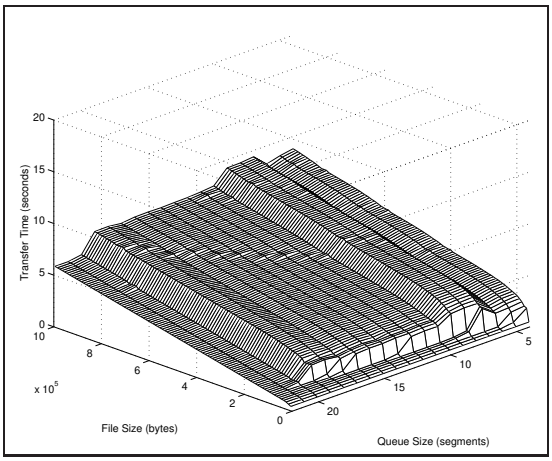

Figure 3: TCP_2 (second proposed change).

how a particular TCP version behaves. In addition, researchers may want to measure some of the following quantities.

- **Loss Rate.** The number of segments dropped by the network is an indicator of the aggressiveness of TCP (or an extension) in a given scenario. Depending on how TCP measurements are taken (see below) the loss rate may not be available, but may be approximated by using the number of retransmissions, rather than the number of drops. Note that the throughput is often highly correlated with the loss rate (i.e., the higher the loss rate the lower the performance). However, the correlation does not always hold. For example, the throughput of small files is often more dependent on the slow start algorithm than on the loss rate.

- **Fairness.** The impact a TCP connection has on the other traffic sharing the network path should also be assessed, if possible (easy in a simulator, but hard in tests over the Internet). The exact definition of fairness and, therefore how to measure it, is somewhat fuzzy. Loosely, a "fair TCP" does not steal bandwidth from competing flows. For instance, consider a TCP flow, F, that never reduces its congestion window. When the network becomes congested competing TCP flows will back-off, while flow F will not. Thus, flow F is stealing bandwidth from the competing flows and using a disproportionate amount of the bandwidth. Several methods for measuring TCP's fairness to competing traffic have been used in various studies [Jai91, BP95, All99].

- **Router Queue Length.** Some TCP experiments can benefit from measurements of the router queue length (again, easy in a simulator but non-trivial in Internet tests). The length of the router queue can have an impact on the end-to-end delay seen by an application. For instance, interactive applications may call for an end-to-end delay as low as possible, while end-to-end delay probably doesn't matter as much for bulk data transfers.

- **Other Measurements.** While the above list of metrics is a good start, there are many other ways to assess the behavior of TCP, which may be of particular benefit to a given experiment. For instance, some extensions might cause an increase in burstiness, which should be quantified. Meanwhile, other studies might involve measuring the time a TCP connection spends waiting for the retransmission timer to expire (e.g., [AP99]). Therefore, we recommend that researchers think about exactly what parts of TCP are being changed and what measurements might make sense when determining what data to collect and what measurements to distill from that data.

Next, the researcher must decide how to obtain the data from which the above quantities can be derived. One popular method for measuring the performance of TCP is to measure throughput at the application level (most FTP applications report throughput after a file has been transfered, for example). However, for very short files the application-reported numbers may be inaccurate in many cases. One problem is that an application can simply hand an entire small file to the operating system, which buffers it for transmission. From the point of view of the application, all the data is sent and so the timer that is measuring the transfer length is stopped and an overestimate of the throughput is reported. A contrast between the throughput obtained from an FTP application and the throughput obtained from packet-level traces is given in [Ric99]. Also note that the application cannot gather much beyond throughput measurements (e.g., number of segment drops, etc.).

Given the problems with application reported performance, we recommend researchers use packet-level trace files to derive the measurements (in simulation or live testbed/Internet experiments). The "vantage point" from which the trace is taken is also important. For instance, a trace taken on the same machine that is sending the data will see all the packets that are sent regardless of whether they are dropped by intermediate routers. On the other hand, a trace taken at the receiver can be used to figure out exactly which segments successfully arrived and which were dropped by the network. Therefore, we recommend taking packet-level trace files on both the sending and receiving sides of the network path, if possible, as such measurement provides the most complete set of data. Such measurements can be somewhat complicated to analyze, as explained in [Pax97a].

Finally, we note that in most cases we recommend that researchers take packet-level traces on the hosts involved in the TCP transfer. Alternatively, packet-level traces can be taken on a nearby host that can watch the traffic to and from the endpoint of the TCP connection. However, taking traces on a second machine can be somewhat problematic. For instance, the trace host may see a packet that the TCP endpoint does not. Or, the trace host may see a packet sooner than the TCP endpoint, hence skewing the transfer time. On the other hand, if the TCP endpoint is a busy machine (generating many TCP connections, for instance) it may not have the resources needed to also obtain an accurate packet-level trace. In this case, another host on the same local area network can be used to take the packet-level trace without too many problems (in general).

11 Additional Considerations

Several non-TCP elements of a network can impact the results of TCP experiments. We recommend that researchers give particular thought to the following elements of the network. And, in general, researchers should think about all aspects of the network path under their control (i.e., everything in simulation and probably only the TCP settings in live Internet tests).

- **Segment size.** The segment size used by the sending TCP host has a direct impact on the performance obtained [MSMO97]. Using common segment sizes (e.g., 1500 byte Ethernet packets) is generally safe. Also, using Path MTU Discovery [MD90, MDM96] is encouraged, in order to utilize the maximum possible segment size across a given network path (although, Path MTU Discovery can also have a negative impact on throughput, as discussed in [AGS99]).

- **Queues.** The maximum queue length and queueing discipline implemented in the routers along the network path also have an impact on TCP performance. A good rule of thumb for the maximum queue length is at least that given in equation 1, where B is the bandwidth of the channel for which the queue is assigned and R is twice the propagation delay of the channel for which the queue is assigned. The queueing discipline used should reflect the currently implemented and used disciplines in the Internet. Therefore, drop-tail queues (the TCP_c of queueing) and increasingly Random Early Detection [FJ93, BCC$^+$98] queues (the TCP_f of queueing) should be used in general TCP studies.

- **Synchronization.** Several studies have shown how experiments can become synchronized thus biasing the results. For instance, [MSMV99] shows the synchronization of data transmission based on the delayed acknowledgment timer. When such synchronization is noticed we suggest researchers either take steps to remove the synchronization from the experiments, or explain how the effect can happen "in the wild" and the implications of such synchronization. Random telnet traffic (based on *tcplib* [DJ91], for instance) can sometimes be useful to remove synchronization from experiments.

- **Link Layer.** Researchers should pay particular attention to the physical link under their control. Some link layer technologies can be optimized for certain situations and therefore can have a large impact on TCP measurements. For instance, an FDDI ring can be tuned for low latency, however should be tuned for high performance when running TCP experiments. Also, half-duplex Ethernet channels have been shown to exhibit a "capture effect," whereby TCP performance suffers because the data segments are competing for bandwidth with the ACK segments. Finally, there can be many interactions with ATM link layers that can negatively impact TCP performance measurements. We recommend that researchers become very familiar with the particular link layer being utilized and discuss the impact link characteristics have on the results obtained when reporting their findings.

12 Conclusions

As outlined in this paper, there really is not a single "best method" for experimenting with TCP. However, poorly constructed experiments and failure to complete a comprehensive analysis of the results lead to research that is *not* taken seriously by the research and standards communities. While there is not a simple set of steps researchers can take to ensure they produce quality research, the following items, along with the discussion in this paper, provide a starting point. Note that simply following these steps does not guarantee good results. Assumptions, test inputs, and conclusions all must bear close scrutiny because of the diversity of TCP implementations and network environments found in the Internet. However, the list below represents the most significant factors that need consideration and specification in TCP research.

- Choose your environment wisely. Generally, a combination of simulation, testbed tests (possibly with emulators) and live Internet tests is needed to provide good evidence about a particular change to TCP.

- In general, all sending TCPs should include slow start, congestion avoidance, fast retransmit, fast recovery, a SACK-based loss recovery strategy and the Nagle algorithm. All TCP receivers should support delayed ACKs.

- The advertised window used should be large enough, such that the network dictates TCP's behavior, rather than a host limitation.

- The data transmitted and the applications employed should be chosen carefully based on real network traffic patterns. Additionally, realistic competing traffic should be transmitted in the network under test.

- Be aware of your MTU, whether you are using MTU discovery and consider how that may impact your results.

- Everything under the researcher's control should be examined and all configurations should be considered before running experiments.

The hope for this paper is not that it discourages future research into TCP's performance and extensions to TCP, but rather that researchers carefully consider their experiments such that the results are compelling and have a positive impact on the continuing evolution of TCP.

Acknowledgments

We thank all the researchers who have helped us understand TCP and the proper way to evaluate TCP performance over the years. Also, the CCR reviewers, as well as the following people provided valuable feedback on early versions of this paper: Ted Fabian, Sally Floyd, Craig Partridge, Jason Pugsley, Jeff Semke and Tim Shepard. Our thanks to all!

References

[ADLY95] Jong Suk Ahn, Peter Danzig, Zhen Liu, and Limin Yan. Evaluation of TCP Vegas: Emulation and Experiment. In *ACM SIGCOMM*, 1995.

[AFP98] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window, September 1998. RFC 2414.

[AGS99] Mark Allman, Dan Glover, and Luis Sanchez. Enhancing TCP Over Satellite Channels Using Standard Mechanisms, January 1999. RFC 2488.

[All97a] Mark Allman. Fixing Two BSD TCP Bugs. Technical Report CR-204151, NASA Lewis Research Center, October 1997.

[All97b] Mark Allman. Improving TCP Performance Over Satellite Channels. Master's thesis, Ohio University, June 1997.

[All98] Mark Allman. On the Generation and Use of TCP Acknowledgments. *Computer Communication Review*, 28(5), October 1998.

[All99] Mark Allman. TCP Byte Counting Refinements. *Computer Communication Review*, 29(3), July 1999.

[AP99] Mark Allman and Vern Paxson. On Estimating End-to-End Network Path Properties. In *ACM SIGCOMM*, September 1999. To appear.

[APS99] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP Congestion Control, April 1999. RFC 2581.

[Arc] Internet Traffic Archive. http://ita.ee.lbl.gov.

[BCC+98] Robert Braden, David Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and Lixia Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet, April 1998. RFC 2309.

[BLFN96] Tim Berners-Lee, R. Fielding, and H. Nielsen. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.

[BP95] Lawrence Brakmo and Larry Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[BP96] Lawrence Brakmo and Larry Peterson. Experiences with Network Simulators. In *ACM SIGMETRICS*, 1996.

[Bra89] Robert Braden. Requirements for Internet Hosts – Communication Layers, October 1989. RFC 1122.

[DJ91] Peter Danzig and Sugih Jamin. tcplib: A Library of TCP/IP Traffic Characteristics. Technical Report CS-SYS-91-01, University of Southern California, October 1991.

[Fal99] Kevin Fall. Network Emulation in the Vint/NS Simulator. Technical report, University of California, Berkeley, January 1999.

[Fey85] Richard Feynman. *Surely You're Joking Mr. Feynman!*, chapter Cargo Cult Science, pages 308–317. Bantam, 1985.

[FF96] Kevin Fall and Sally Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[FGM+97] R. Fielding, Jim Gettys, Jeffrey C. Mogul, H. Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, January 1997. RFC 2068.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. *Computer Communications Review*, 24(5):10–23, October 1994.

[Flo99] Sally Floyd. Validation Experiences with the NS Simulator. Technical report, ACIRI, April 1999. White paper submitted to the DARPA Workshop on Validation of Large Scale Network Simulation Models.

[Hei97] John Heidemann. Performance Interactions Between P-HTTP and TCP Implementations. *Computer Communication Review*, 27(2):65–73, April 1997.

[Hey90] A. Heybey. The Network Simulator. Technical report, MIT, September 1990.

[HK99] Tom Henderson and Randy Katz. Transport Protocols for Internet-Compatible Satellite Networks. *IEEE Journal on Selected Areas of Communications*, 1999. To appear.

[Hoe96] Janey Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *ACM SIGCOMM*, August 1996.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. Wiley, 1991.

[JBB92] Van Jacobson, Robert Braden, and David Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[Kes88] Srinivasan Keshav. REAL: A Network Simulator. Technical Report 88/472, University of California Berkeley, 1988.

[KL86] B. Kantor and P. Lapsley. Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News, February 1986. RFC 977.

[MA99] Matt Mathis and Mark Allman. Empirical Bulk Transfer Capacity, June 1999. Internet-Draft draft-ietf-ippm-btc-framework-01.txt.

[Mah99] Jamshid Mahdavi. Enabling High Performance Data Transfers on Hosts. Technical report, Pittsburgh Supercomputer Center, June 1999. http://www.psc.edu/networking/perf_tune.html.

[MD90] Jeffrey C. Mogul and Steve Deering. Path MTU Discovery, November 1990. RFC 1191.

[MDM96] Jack McCann, Steve Deering, and Jeffrey C. Mogul. Path MTU Discovery for IP version 6, August 1996. RFC 1981.

[MF95] Steven McCanne and Sally Floyd. NS (Network Simulator), 1995. URL http://www-nrg.ee.lbl.gov.

[MM96a] Matt Mathis and Jamshid Mahdavi. Diagnosing Internet Congestion with a Transport Layer Performance Tool. In *Proceedings of INET*, June 1996.

[MM96b] Matt Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *ACM SIGCOMM*, August 1996.

[MM96c] Matt Mathis and Jamshid Mahdavi. TCP Rate-Halving with Bounding Parameters. Technical report, Pittsburgh Supercomputer Center, October 1996. URL: http://www.psc.edu/networking/papers/FACKnotes/current/.

[MMFR96] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. RFC 2018.

[Mog92] Jeffrey C. Mogul. Observing TCP Dynamics in Real Networks. In *ACM SIGCOMM*, pages 305–317, 1992.

[MSML99] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Kevin Lahey. The Rate-Halving Algorithm for TCP Congestion Control, August 1999. Internet-Draft draft-mathis-tcp-ratehalving-00.txt (work in progress).

[MSMO97] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27(3), July 1997.

[MSMV99] Greg Minshall, Yasushi Saito, Jeffrey C. Mogul, and Ben Verghese. Application Performance Pitfalls and TCP's Nagle Algorithm. In *Workshop on Internet Server Performance*, May 1999.

[Nag84] John Nagle. Congestion Control in IP/TCP Internetworks, January 1984. RFC 896.

[PAD+99] Vern Paxson, Mark Allman, Scott Dawson, William Fenner, Jim Griner, Ian Heavens, Kevin Lahey, Jeff Semke, and Bernie Volz. Known TCP Implementation Problems, March 1999. RFC 2525.

[Pax97a] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM*, September 1997.

[Pax97b] Vern Paxson. End-to-End Internet Packet Dynamics. In *ACM SIGCOMM*, September 1997.

[Pax97c] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. Ph.D. thesis, University of California Berkeley, 1997.

[PBS+96] Venkata Padmanabhan, Hari Balakrishnan, Keith Sklower, Elan Amir, and Randy Katz. Networking Using Direct Broadcast Saellite. In *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*, November 1996.

[PF95] Vern Paxson and Sally Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[PF97] Vern Paxson and Sally Floyd. Why We Don't Know How to Simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[PMAM98] Vern Paxson, Jamshid Mahdavi, Andrew Adams, and Matt Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications*, 1998.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

[PR85] Jon Postel and Joyce Reynolds. File Tranfer Protocol (FTP), October 1985. RFC 959.

[PS98] Steve Parker and Chris Schmechel. Some Testing Tools for TCP Implementors, August 1998. RFC 2398.

[RF99] K. K. Ramakrishnan and Sally Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP, January 1999. RFC 2481.

[Ric99] Alan Richard. ISINT Performance Validation Test Report. Technical Report CR-209407, NASA Glenn Research Center, June 1999.

[Riz97] Luigi Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *Computer Communications Review*, 27(1):31–41, January 1997.

[SMM98] Jeff Semke, Jamshid Mahdavi, and Matt Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM*, September 1998.

[Tec] Third Millennium Technologies. OpNet Modeler. http://www.mil3.com.

[TMW97] Kevin Thompson, Gregory Miller, and Rick Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, November/December 1997.

[VH97] Vikram Visweswaraiah and John Heidemann. Improving Restart of Idle TCP Connections. Technical Report 97-661, University of Southern California, August 1997.