SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments

Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble
University of Washington

Abstract— **The measurement of network bandwidth has become increasingly important to many Internet services and systems, including the creation of robust overlays for peer-to-peer systems or application-level multicast trees, the selection a server among multiple server mirrors, and the placement of server replicas to improve the performance of content distribution networks. To be useful in practice, a bandwidth estimation tool must be accurate, fast, scalable, and able to work in uncooperative environments. This paper presents the design, implementation, and evaluation of SProbe, a bottleneck bandwidth estimation tool that possesses all these practical properties. SProbe exploits properties of the TCP protocol in a manner similar to the "Sting" tool. As part of the evaluation of SProbe, we performed a study of over 50,000 uncooperative wide-area hosts comprising a variety of operating systems and networks, which demonstrates its accuracy, scalability, speed, and most importantly, its practicality. Finally, we report on our experience of performing Internet measurements using SProbe, and we make recommendations for future protocol and network designs.**

Keywords— **Network measurements, Bottleneck bandwidth, Available bandwidth, Peer-to-Peer, Server selection, Server replica placement.**

I. INTRODUCTION

MEASURING network bandwidth is important to wide-area Internet services and distributed systems. Knowledge of network bandwidth can improve these systems in many ways, such as creating more robust overlays for peer-to-peer systems [1], [2], [3], [4] or application level multicast trees [5], dynamically selecting a server among several mirrors [6], and dynamically placing server replicas to improve the performance of content distribution networks [7], [8].

The precise meaning of "network bandwidth" can be defined according to several metrics, including:

- throughput – the number of transferred bytes over a network path during a fixed amount of time.
- available bandwidth – the maximum attainable throughput of a newly started flow over a network path.
- bottleneck bandwidth – the maximum throughput that is ideally obtained across the slowest link of a network path.

Measuring throughput and available bandwidth is complex in practice, since they depend on many different factors, including latency, loss rate, network path load, and TCP implementation details. These factors are not only hard to measure, but they also fluctuate over short periods of time and the interactions among them are hard to predict. In contrast, algorithms and techniques for measuring and estimating the bottleneck bandwidth of a network path are well understood (at least, theoretically).

To be useful in the context of Internet services and distributed systems, we believe that a practical bottleneck bandwidth estimation tool must be able to work in an **uncooperative environment**, i.e., one in which measurement software is only deployed locally on the measurement host. In addition, because modern services and systems are intended to span a very large number of hosts and network paths, we believe that a practical tool must also produce **accurate** and **fast** estimates using little amounts of probing data, so that it can **scale** to a large number of estimates. As this paper will show, none of the current tools have all of these properties.

The main contribution of this paper is the design, implementation, and evaluation of SProbe, a fast, scalable, and accurate measurement tool that works in uncooperative environments. SProbe exploits properties of the TCP protocol in a manner inspired by by Savage's *Sting* tool [9]. Our evaluation reports on the experience of using SProbe in a study of over 50,000 uncooperative hosts comprising a wide variety of host operating systems, network connections, and geographic locations; to the best of our knowledge, no previous tool or technique has been evaluated on such an extensive scale. This study directly confirms the practicality of our tool.

A second contribution of this paper is a qualitative comparison of a representative set of modern bottleneck bandwidth estimation tools. As part of this comparison, we identify the key properties of such tools that influence their effectiveness when used in the context of modern systems. The results of this comparison show that of no current tool possesses the right combination of these required properties. However, we are delighted to report that based on our discussions with the authors of Nettimer [10], our techniques have been recently incorporated into their tool.

The rest of the paper is structured as follows. Section II presents a qualitative analysis of the properties of bottleneck bandwidth tools along with their strengths and weaknesses. Section III describes the application of bandwidth estimation in the context of several current research problems and compiles a list of desired properties that an estimation tool must have. The architecture of SProbe is presented in Section IV. Section V presents an evaluation of SProbe. Section VI contains some reflections on using SProbe in a large-scale measurement project. Finally, we present conclusions and future work in Section VII.

II. A Qualitative Analysis of the Properties of Bottleneck Bandwidth Tools

Estimating bottleneck bandwidth is not a new area of research [11], [12], [13], [14], [15], [16], [17]. Broadly speaking, all existing techniques fall into two categories: the “one-packet” technique and the “packet pair” technique. Several tools that use these techniques have been developed and studied in the past [10], [18], [19], [20], [21], [22]. In this section of the paper, we present a qualitative analysis of key properties of these tools. First, we introduce some terminology.

The endpoints of a network path are referred to as *endhosts*. A *cooperative endhost* is an endhost on which measurement software has been deployed, and an *uncooperative endhost* is one on which measurement software cannot be deployed. However, an uncooperative endhost can still unwittingly participate in a measurement; for example, content served by a Web server can be used as input traffic to a measurement tool.

A measurement is done in a *cooperative environment* when the measured network path runs between two cooperative endhosts. In contrast, when one endhost is uncooperative, the measured is performed in a *uncooperative environment*. In an uncooperative environment, we will sometimes refer to the cooperative endhost as the *local endhost*, as opposed to the *remote host*, the uncooperative one.[1]

A network path is *symmetric* if the bottleneck bandwidths in each direction are equal, otherwise the path is *asymmetric*. In an uncooperative environment, the path direction from the cooperative endhost to the uncooperative endhost is referred to as the *downstream* direction, in contrast to the *upstream* direction, from the uncooperative to the cooperative endhost.

Finally, *cross-traffic* represents existing network traffic that causes queueing delays of the probing packets.

[1] We are not aware of any general technique that measures bottleneck bandwidth between two uncooperative endhosts.

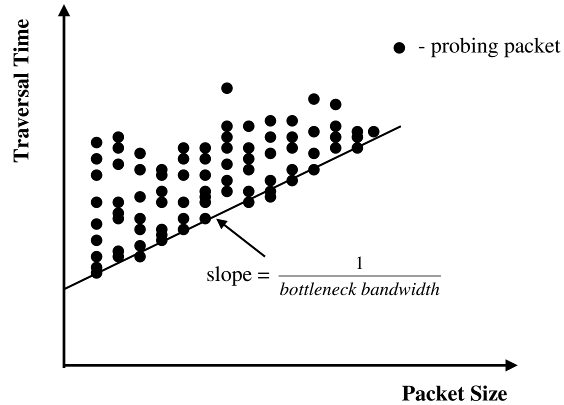

Fig. 1. **One-Packet Model: ideally, the minimum value of the traversal times for each packet size approximates a line whose slope is the inverse of the bottleneck bandwidth.**

A. *The One-Packet Technique*

The one-packet technique, first described in [23], relies on the observation that a packet's traversal time across a single link can be approximated by the sum of three variables: queueing delay, transmission delay (signal modulation and serialization), and signal propagation delay across the transmission medium. In the absence of cross traffic, queueing delay can be assumed to be negligible; by taking a large number of measurements, it is likely that at least one will be obtained without queueing delay.

A second assumption is that transmission delay varies linearly with packet size and can be approximated by the ratio of packet size to link bandwidth. A final assumption of the model is that latency remains constant for different packet sizes. Therefore, sending a large number of packets of different sizes ensures that the minimum value of their transmission times will approximate a line whose slope is the inverse of link bandwidth (Figure 1). This technique produces an estimate at each hop along the measured network path; the bottleneck bandwidth of the entire path is the minimum value of these link bandwidths. For more information, see [16], [23], [24].

A.1 Strengths and Weaknesses of the One-Packet Technique and Tools

Several tools are based on the one-packet model, including *Pathchar* [18], *pchar* [19], and *clink* [20]. An attractive property of the one-packet model is its ability to measure the bottleneck bandwidth in an uncooperative environment. Also, this technique produces a picture of the entire network path measured by estimating the link bandwidths at each hop along the path. Finally, because they send a large amount of probing packets and record the minimum traversal times, the one-packet tools are less susceptible to cross traffic.

Unfortunately, the one-packet model has several important limitations. First, the current tools implementing this technique rely on a functional ICMP implementation at each router along the measured network path. Second, this technique measures bandwidth in a single direction, from the local to the remote endhost. Third, because the estimates of each link rely on the estimates of the previous links, the errors accumulate and amplify with each measured link. (Previous studies have presented evidence that these tools are inaccurate and slow [16], [17].) Finally, the large number of probing packets generated by the tools adds considerable stress to the network path. A quick calculation reveals that for a single Ethernet hop with a latency of 1ms, the average bandwidth consumed is 6.02Mbps [25]. There are several negative implications due to this *network flood*: the tools are *unscalable*, *slow* and *inflexible to bandwidth changes*.

B. The Packet Pair Technique

The packet pair technique was introduced by [11] and [12]. Two consecutive packets are queued one after the other at the bottleneck queue of a network path, hence the name *packet pair*. After traversing the bottleneck link, the time dispersion between the two packets is linearly related to the bottleneck link bandwidth, assuming same-sized packets. For this technique to work, the packets must be large enough so that queuing occurs at the bottleneck link. After the bottleneck link, the temporal spacing between the packets will remain constant, as the packets have the same size and no other link has a bandwidth lower than the bottleneck link. Figure 2 illustrates the technique. For more details on this technique, see [15], [16].

Four assumptions must be valid for this technique to work in practice [16]. First, the packets' size must be large enough so that they queue at the bottleneck link but no later link. Second, the two probe packets must be sent consecutively, with no space between them. Third, the bottleneck router must use FIFO queueing. Finally, transmission delay must be proportional to packet size.

B.1 Strengths and Weaknesses of the Packet Pair Technique and Tools

Unlike the one-packet model, the packet pair technique only produces an estimate of the bottleneck link bandwidth, as opposed to the bandwidth of each link along the network path. Since the bottleneck link estimate does not rely on previous links' bandwidth estimates, the packet pair technique is typically more accurate.

Like the one-packet technique, the accuracy of packet pair measurements can degrade due to cross traffic interference. A rogue packet queued at the bottleneck link between the packet pair invalidates the measurement. Current packet pair tools deal with cross traffic by sending a large amount of probe packet pairs and gathering a large number of measurements. Assuming that cross traffic interference does not dominate the measurements, statistical algorithms are used to extract the average, free-of-cross-traffic case. Different statistical algorithms have been proposed to extract accurate estimates from sets of gathered data points [14], [26], [17].

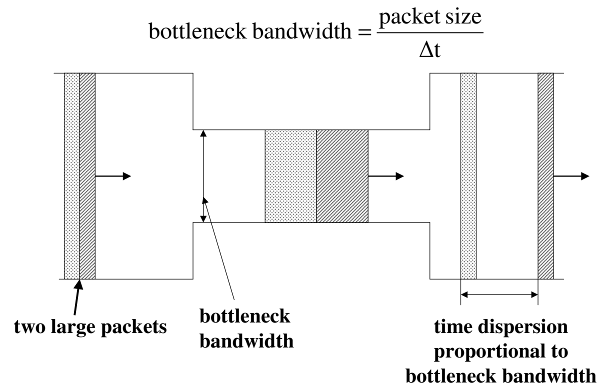

Fig. 2. **Packet Pair Model: after the bottleneck link, the time dispersion between the packets is proportional to the bottleneck bandwidth.**

Several implementations of the packet pair technique exist: *bprobe/cprobe* [21], *Nettimer* [10], *pathrate* [22]. *Nettimer* is a passive tool that relies on monitoring traffic between the endhosts, whereas *bprobe/cprobe* and *pathrate* actively send probe packets. Estimates can be based on different protocols: *bprobe/cprobe* uses ICMP, *Nettimer* uses TCP and *pathrate* uses UDP.[2] *bprobe/cprobe* is designed for uncooperative environments, whereas *pathrate* assumes a cooperative environment. *Nettimer* has different estimating techniques for both cooperative and uncooperative environments.

III. On the Suitability of Current Tools

This section of the paper presents three existing applications that would strongly benefit from fast and scalable bandwidth estimation. Currently, the deployed instances of these applications are not bandwidth estimation aware, and as such, they constitute an uncooperative environment. Motivated by these applications, we extract a set of properties important to the practicality of bandwidth estimation tools. We analyze current tools with respect to these properties; a summary of this comparison is in Table I.

[2] *pathrate* also exchanges control information between the endhosts over a TCP connection, however the probes are UDP packets.

Peer-to-peer overlay formation: Many current peer-to-peer systems [27], [28], [1], [2], [3], [4] consist of independent, mutually distrustful hosts that self-organize into an application-level overlay over which data is exchanged. These systems would benefit from a tool that provides *accurate* and *fast* estimates of the bottleneck bandwidth in *uncooperative environments*: if the system can discover peers' bandwidths, it can form a more efficient and scalable overlay. For example, a system might choose to cluster high-bandwidth peers together, delegate system responsibilities to higher-bandwidth peers, or shed load from the lower-bandwidth peers.

Current peer-to-peer systems often rely on users to report their bandwidths; people use reported bandwidth to decide where to download a piece of content. However, in [29], the authors show that a substantial fraction of hosts choose to misrepresent or not to report their bandwidths. A fast and accurate tool that works in uncooperative environments would remove the assumption that users in the system are trustworthy.

Application-level multicast tree formation: Current designs for end system multicast have proposed algorithms for constructing multicast overlay trees optimized with respect to both latency and bandwidth [5]. These designs constantly monitor the latencies and bandwidths between different multicast members by actively probing their network paths. Such a system can benefit from a bandwidth estimation tools that is *fast* and *scalable*.

Wide-area, large-scale network measurements: One way of understanding the performance and properties of current network applications and protocols is performing wide-area, large-scale network measurements. Given the explosion in the number of Internet hosts and networking infrastructure, it has become standard practice to measure the characteristics of tens of thousands of network paths and endhosts. In addition to the *accuracy* of the tool, *speed* and *scalability* become as important when dealing with such a large number of measurements.

A. Properties of the Ideal Bottleneck Bandwidth Tool

To be useful for the scenarios presented above, a bottleneck bandwidth estimation tool must have four vital properties: accurate and fast measurements, the ability to work in uncooperative environments, and scalability. Fast measurements are essential when dealing with multiple network paths and endhosts; spending a minute on a single measurement can make a tool too slow to be valuable in practice.[3] In measurements dealing with tens of thousands of hosts, deploying software on all endhosts is impossible in practice; even a low degree of cooperation, such as having enough data to transfer between the endhosts, is sometimes an infeasible assumption to make. A tool should be able to cope with tens of thousands of network paths and Internet hosts, implying that tools that need to flood the network to make a single measurement simply will not scale.

Tools that base their estimates on passively monitoring are intrinsically scalable since they do not consume any extra bandwidth. However, passive tools suffer from a basic problem that makes them infeasible under realistic scenarios: existing traffic might not be suitable for the tool to produce an estimate. The packet pair technique requires that large (700-1500 bytes) consecutive packets are exchanged between the endhosts. Therefore, a tool should *actively* generate the appropriate probing traffic in order to estimate bandwidth accurately.

In addition, the protocol a tool uses to make its estimate can also be important. Given the current state of the Internet, we believe that a tool should not use ICMP, since ICMP packets are often filtered, dropped, or even answered by an ISP on behalf of its clients. Also, several operating systems (e.g., Solaris) limit the rate of ICMP responses [30]. Instead, we believe that a practical tool should use *TCP* or *UDP* packets.

A good way to understand a tool's weaknesses and limitations is to test it in a variety of controlled, reproducible experiments. Only then can the accuracy of the tool's estimates be validated against known, true values. In contrast, validating experiments on a real, complex network (such as the Internet) is extremely difficult. Currently, the only packet pair tools that have been validated in a controlled environment are *Nettimer* and *pathrate*, which were evaluated using a network simulator. However, these evaluations were for long flows, and the accuracy of these tools for short flows (as are typical with many Internet protocols) has not yet been validated.

A tool should be flexible to bandwidth changes. Unfortunately, current tools strive to achieve flexibility at the cost of highly fluctuating estimates. Instead, we believe that a tool can be flexible to bandwidth changes by producing fast estimates, in a matter of milliseconds. In this way, in a highly mobile environment, an application can determine how frequent the tool should run to capture the changes in the network paths.

Finally, an ideal tool should be resilient to cross traffic. A tool should either prevent the cross traffic from degrading its accuracy, or detect its presence and choose not to report a potentially inaccurate estimate.

Given the outline of these properties, we can now an-

[3] A minute per measurement translates to spending a week for about 10,000 sequential measurements

TABLE I

THE DESIRED PROPERTIES OF AN IDEAL TOOL (THE FIRST FOUR PROPERTIES ARE VITAL). A "$\star$" REFERS TO A PROPERTY THAT IS ONLY PARTIALLY ACHIEVED BY A TOOL.

Desired Properties	*Pathchar*	*pchar*	*clink*	*bprobe*	*pathrate*	*Nettimer*	*SProbe*
Accurate	✓	✓	✓	✓	✓	✓	✓
Fast							✓
Works in Uncooperative Environments	✓	✓	✓	✓		$\star$	✓
Scalable						✓	✓
TCP/UDP Based					✓	✓	✓
Active	✓	✓	✓	✓	✓		✓
Resilient to Cross-Traffic	✓	✓	✓	✓	✓	✓	$\star$
Works on Asymmetric Paths					✓	✓	✓
Flexible to Bandwidth Changes				✓		✓	✓
Controlled Evaluations					✓	✓	

swer the following question: *which properties do the current tools possess?* To answer this question, we analyzed the properties of six popular tools: *Pathchar*, *pchar*, *clink*, *bprobe*, *pathrate*, *Nettimer*. The results are summarized in Table I. As this table shows, none of the current tools possesses all the desired properties, let alone all four vital properties outlined at the beginning of this section. In the next section of this paper, we present the design of SProbe, a tool that possesses all four of the vital properties.

IV. ARCHITECTURE OF SPROBE

Since the one-packet technique floods the network to produce a single estimate, SProbe's architecture is based on the packet pair technique. Furthermore, SProbe is designed to be deployed at the single, uncooperative endhost.

A. Measuring Bottleneck Bandwidth in the Downstream Direction

To measure downstream bottleneck bandwidth, a large packet pair needs to traverse the path from the local to the remote host. Due to the nature of the uncooperative environment, it is practically impossible to know the time dispersion of the packet pair arriving at the remote host. However, in certain cases, the remote host answers the received packets. Assuming that the packets sent as answers do not queue at the bottleneck bandwidth on the reverse path, their time dispersion can be used as an approximation to the initial, large packet pair time dispersion.

SProbe relies on an exploitation of the TCP protocol, similar to the ones described in [9] to perform the sequence of packet exchanges described above. In the TCP protocol, a SYN packet pair sent to an inactive port of the remote machine is answered by a RST packet pair. Regular SYN packets are 40-byte packets, having no payload data. SProbe appends a large payload (1460 bytes) to each sent SYN packet. However, the answered RST packets are small packets (40 bytes), which are unlikely to queue at the bottleneck link on the reverse path. On the local host, SProbe uses the time dispersion of the received RST packet pair as an approximation to the remote host time dispersion of the SYN packet pair. Note that this packet exchange only relies on a correct TCP implementation at the remote host and is sufficient to measure the downstream bottleneck bandwidth. Figure 3 illustrates the packet exchange initiated by SProbe.

On certain network links (e.g. modem lines), the packets are compressed before traversing the link. This effect can alter packet size and impact the accuracy of the packet pair measurement. Thus, SProbe appends a precompressed payload to the SYN packets to minimize the compression effects on the packet size. We are not aware of any other tool that explicitly deals with packet compression. Finally, although the current implementation of SProbe sends SYN packets, the same measurement sequence can be achieved using FIN, ACK or DATA packets.

Our technique, however, has several limitations. Because firewalls silently drop SYN packets to inactive ports, SProbe cannot distinguish hosts behind firewalls from offline hosts. When packet loss occurs, SProbe will timeout after five seconds and terminate with an *unanswered* probe message. It is up to the user or the application invoking SProbe to decide whether to retry the measurement.

Previous active tools send many probing packets to decrease the likelihood of cross traffic, reordering or multichannel links interference. Unfortunately, this approach has two inherent drawbacks – (1) the tool is *unscalable* and (2) the tool is *slow*. Instead, SProbe relies on a small amount of data to remain resilient in the face of these ad-

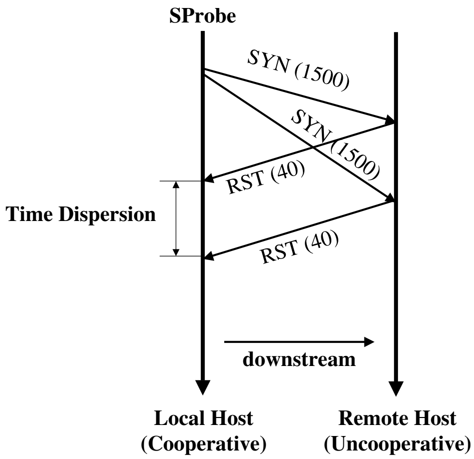

Fig. 3. **The packet exchange initiated by SProbe to estimate the bottleneck bandwidth in the downstream direction.**

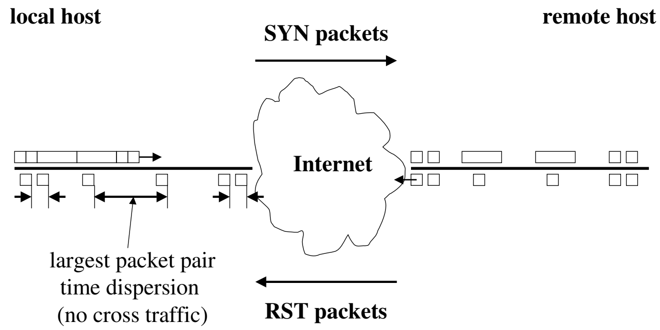

Fig. 4. **The consistent arrival times heuristic test. The time dispersion of the RST packets corresponding to the large SYN packet pair (the middle packets) should be the largest dispersion among packet pairs when no cross traffic is present.**

verse effects. SProbe shapes its probing traffic to reveal information about the conditions of the measured network path. It sends a train of small, 40-byte SYN packets in the middle of which a large packet pair (1500-byte) is placed. The remote endhost answers with a same size train of RST packets. SProbe analyzes the time dispersion of the packets in the train and uses two heuristics that determine the presence of cross traffic, reordering or multi-channel links. SProbe currently uses a train of six SYN packets, two of which constitute the large packet pair.

The Shuffle Heuristic Test: SProbe uses the sequence number of the RST packets to determine whether the ordering of the received packets has been preserved relative to the ordering of the sent SYN packets. When packets were reordered, SProbe discards the measurement and returns an *unknown* estimate. Note that a multi-channel link is also likely to reorder the train packets.

The Consistent Arrival Times Heuristic Test: When the RSTs are not reordered, the time dispersion of the two RSTs in the middle of the train should be larger than the dispersion of any of the smaller 40-byte packet pairs. If not, it is likely that a rogue packet was placed between 40-byte packet pair. This indicates cross-traffic presence during probing; SProbe discards the measurement and returns an *unknown* estimate. Figure 4 illustrates the packet train exchange and the consistent arrival times heuristic test.

However, during our evaluation, the estimates based on measurements eliminated by the consistent arrival times heuristic test are within a factor of two of the estimates based on measurements that passed this test. We believe that only an experiment under controlled conditions will validate the effect and the selectivity of this heuristic test. We plan to address this issue in future work.

B. *Measuring Bottleneck Bandwidth in the Upstream Direction*

Unlike the downstream direction, where virtually no cooperation is required from the remote host, the upstream direction requires a low degree of cooperation, although no measurement software needs to be deployed. SProbe uses a TCP connection to produce an estimate. During slow-start, the TCP protocol consecutively sends a number of packets equal to its congestion window. Whenever these packets are large, they can serve as probing packets to estimate the upstream bottleneck bandwidth.

This is not a new approach, previous tools rely on consecutive packet pairs to dominate the exchanged traffic over the course of long TCP flows, and use statistical algorithms to ensure they converge to the *normal-case* value. SProbe uses a similar idea with a small degree of fine-tuning, to ensure a rapid estimate. In the case of a Web server, SProbe initiates a TCP connection, advertising an 1500-byte MSS, and sends an application level request for data (e.g. an HTTP GET-request). On receiving the GET-request, the Web server consecutively sends as many data packets as the initial value of its congestion window. Using their arrival time dispersion, SProbe produces an estimate of the upstream bottleneck bandwidth. A recent study shows that most Web servers start with an initial congestion window size of at least two segments [31].

If the initial congestion window size is one, SProbe will timeout after waiting for a packet pair and will acknowledge the packet received. This will cause an increase in the server's congestion window size, and a packet pair is sent. In the case of a packet loss, SProbe acknowledges the first unreceived byte, until the remote host sends a packet pair. Finally, when the packets received are not large,

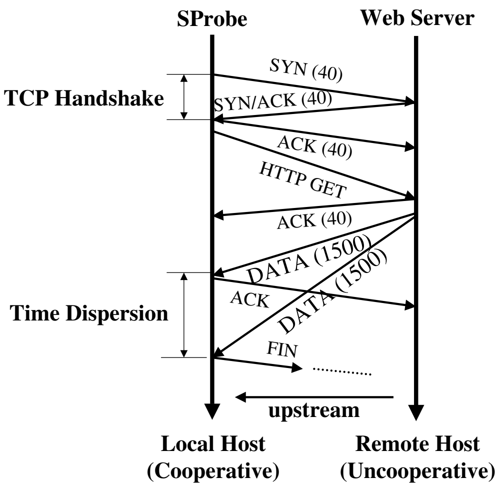

Fig. 5. **The packet exchange initiated by SProbe to estimate the bottleneck bandwidth in the upstream direction.**

SProbe discards the measurement and reports an *unknown* estimate. This is a violation of the packet pair assumptions, and we believe that, in this case, a measurement tool should return an *unknown* estimate, rather than produce a potentially inaccurate one.

Using this technique, SProbe produces *very fast* estimates. Figure 5 shows the typical packet exchange sequence used by SProbe. In general, three round-trip times (RTT) and less than 3KB are sufficient for SProbe to produce an estimate. Previous studies [32], [33] show that the median size of a Web page is between 2.2KB and 4KB. SProbe trades the uncertainty of cross traffic occurance for a fast estimate. Our experiments, presented in Section V, show that 85% of SProbe's estimates are within a factor of two from estimates based on long TCP flows.

V. Evaluation of SProbe

Previous work has presented careful, controlled experiments evaluating the strengths and weaknesses of the packet pair estimation technique [12], [15], [16]. Rather than performing another investigation into the efficacy of the technique itself, we instead focussed on demonstrating the practicality of using SProbe in the context of two wide-area scenarios: (1) a large-scale network measurement project performed at University of Washington, and (2) an instance of the server selection problem in which we explored the selection of a Web server from 248 different mirrors serving the same content, placed in 57 different countries on six different continents.

All results reported were measured on a Dell PowerEdge 6450, dual processor Pentium III 700MHz with 2GB of RAM and an Intel Pro/1000 gigabit Ethernet card, running Linux Red Hat 7.0, kernel version 2.4.0.

A. Large-Scale Network Measurements

A recent project at University of Washington [29] studied two of the most popular file sharing peer-to-peer systems: Napster and Gnutella. This project evaluated characteristics of peers participating in these systems by directly measuring properties such as each peer's bottleneck bandwidth, latency, availability, and extent of content sharing. Over the course of four consecutive days, 546,401 unique Napster IP addresses and 1,180,205 unique Gnutella IP addresses were measured. The project used SProbe to gather bottleneck bandwidth estimates of these peers; SProbe possessed the right combination of properties that enabled it to be used for this project. In particular, its speed and ability to work in uncooperative environments, along with its scalability and accuracy were essential for gathering this large number of measurements. As a result of this project, we have a large amount of data that demonstrates the tool's practicality, efficiency, and accuracy.

Napster's central servers provide the connection bandwidth of any peer as reported by the peer itself. The UW measurement project found that a substantial percentage of Napster peers (as high as 25%) chose not to report their bandwidths. However, the bandwidth reported by the other 75% of the peers could be compared to SProbe's bottleneck bandwidth estimates. Furthermore, since the measurement workstations were connected by a gigabit link to the Abilene network, it is likely that the bottleneck link between the measurement workstations and any peer was the last-hop link to the peer itself. Thus, the measured bottleneck bandwidth should ideally match both the reported bandwidth and also the actual speed of the peer's Internet connection.

The typical Napster and Gnutella user connects to Internet using a dialup modem, cable modem, or DSL connection. By resolving a peer's IP addresses into a DNS name, we can often deduce the type of Internet connection used by that peer. For example, if a peer's DNS name contains the word *dialup*, it is extremely likely that the peer uses a dialup modem to connect to the Internet. Unfortunately, the same Internet connection type can imply different network speeds (e.g., a dialup modem's speed can be anywhere between 300bps and 57.6Kbps). In Figure 6, three different subsets of peers have been measured:

- **dialup**: 282 peers whose names contain the word *dialup*.
- **DSL**: 2270 peers whose names contain the word *dsl*.
- **cable modem**: 10,259 peers whose names end in *home.com*, *shawcable.net* or *cgocable.net*.

As Figure 6 shows, SProbe reported a bottleneck bandwidth of less than 100Kbps for 92% of the dialup modems. Similarly, 91% of the measured DSLs have bottleneck

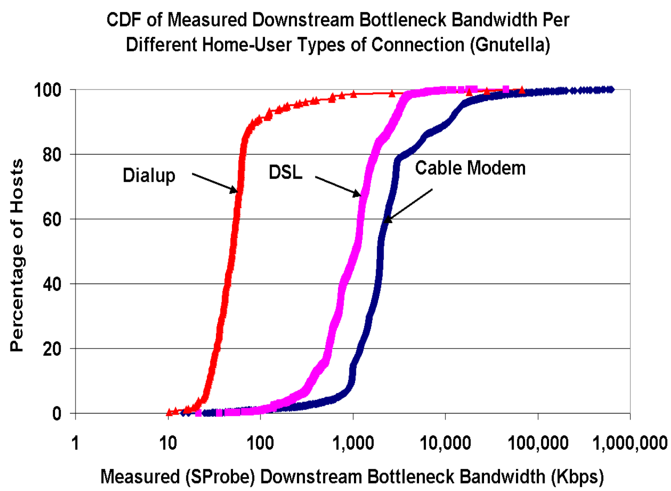

Fig. 6. **Measured downstream bottleneck bandwidths for Gnutella peers, grouped by the types of their Internet connections.**

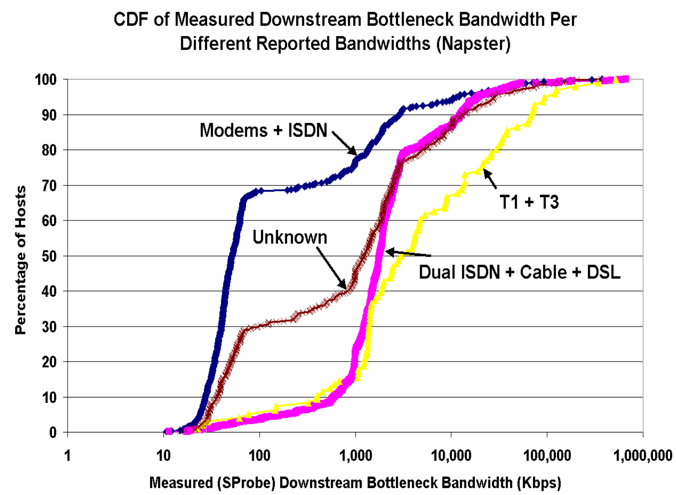

Fig. 7. **Measured downstream bottleneck bandwidths for Napster peers, grouped by their reported bandwidths.**

bandwidth estimates between 128Kbps and 3Mbps, which corresponds to the range of speed supported by DSL modems. For cable modems, it is less clear what their actual connection speed are, since different vendors report downstream speeds of anywhere between 1Mbps to 10Mbps. Only 78% of the cable modems have a measured bottleneck bandwidth within this range; SProbe reports a bottleneck bandwidth of higher than 10Mbps for 11% of them. These results validate the accuracy of SProbe.

Figure 7 presents the cumulative distribution of measured downstream bottleneck bandwidths for Napster peers, grouped according to reported bandwidth. In Napster (and any similar system), peers have an incentive to report a smaller bandwidth than their actual value to discourage others from initiating downloads and consuming network resources. Conversely, we expect few users with high reported bandwidths to have low actual bandwidths, as this would encourage other peers to consume their (scarce) network resources. Indeed, the estimates produced by SProbe confirms this effect. As we see in Figure 7, SProbe estimates a downstream bottleneck bandwidth of more than 100Kbps for 30% of the peers that have reported bandwidths of 64Kbps or less: many peers are in fact reporting lower than actual bandwidths. Conversely, only 9% of the users reporting high bandwidths (3Mbps or higher) are observed to have lower actual bandwidths.

The strong correlations observed between measured bandwidth and deduced connection type (Figure 6) or reported bandwidth (Figure 7) demonstrate the accuracy and effectiveness of SProbe. Moreover, no other measurement tool would have been suitable to the scale of this project, which validates SProbe's practicality.

B. Measuring Bottleneck Bandwidths from 1000 Web Servers

In this section, we explore the tradeoff between the number of measurement packets used by a packet pair tool (which affects the speed at which it can gather measurements) and the accuracy of the resulting estimate. As we will show, even though it uses a small number of measurement packets, SProbe is still able to maintain a high degree of accuracy relative to a tool that is known to be accurate given a large number of packets.

To perform this experiment, we measured the upstream bottleneck bandwidths from 261 different Web mirrors of the *Linux Documentation Project*.[4] Besides SProbe, the only other packet pair tool that was suitable for this uncooperative environment was Nettimer; since Nettimer only produces upstream estimates in uncooperative environments, we focussed only on the upstream measurements.

For each of the web mirrors, we initiated a download of a 1.6MB file and had Nettimer produce an estimate by monitoring the entire download; note that Nettimer could produce a result with less data, but given the known accuracy of Nettimer for long flows, we regard its estimate using this large amount of data as a close approximation of the true upstream bottleneck bandwidth. Of the 261 downloads, we rejected 13 because Nettimer's estimate was lower than the end-to-end throughput implied by the time the download took.

The first few data packets observed during each download were also fed to SProbe to produce its estimate. We

[4] Although there are 360 mirrors listed at `http://mirrors.linuxmall.com/LDP/mirrors.html`, the particular file that we downloaded was available only at 261 of them.

TABLE II

THE NUMBER OF MEASUREMENTS FILTERED BY THE HEURISTICS TESTS. **# SHUFFLED TRIALS** IS THE NUMBER OF TRIALS THAT FAILED THE SHUFFLE HEURISTIC TEST; **# OF UNSHUFFLED BUT INCONSISTENT ARRIVALS** IS THE NUMBER OF TRIALS THAT FAILED DUE TO THE CONSISTENT ARRIVAL TIMES HEURISTIC TEST.

# trials	# successful trials	# shuffled trials	# unshuffled but inconsistent arrivals
149,133	40,238 (27%)	93,588 (63%)	15,307 (10%)

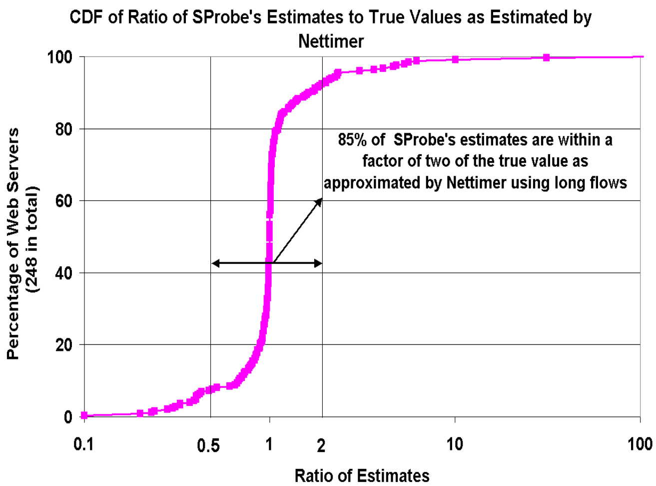

Fig. 8. **CDF of the ratio of upstream bottleneck bandwidths estimates produced by SProbe relative to true values as approximated by Nettimer using a large amount of data.**

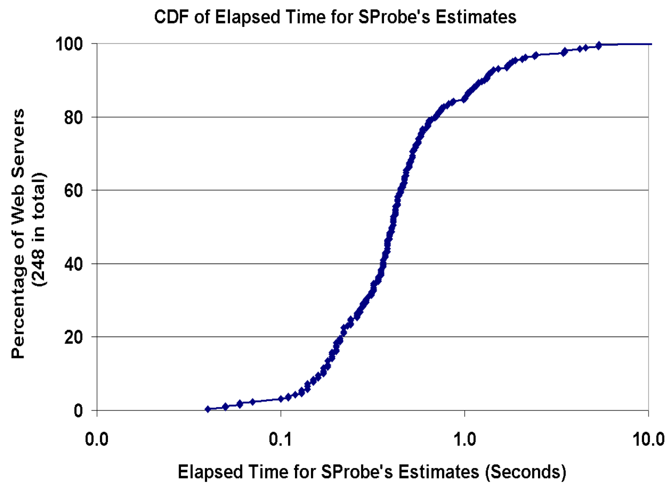

Fig. 9. **CDF of the elapsed times of SProbe's estimates to 248 Web servers.**

then compared the estimates produced by SProbe and Nettimer by taking their ratio. Figure 8 presents the cumulative distribution of this ratio across our measurements. This figure demonstrates that SProbe's accuracy is comparable to that of Nettimer, even though uses a small number of measurement packets: 85% of SProbe's estimates are within a factor of two from Nettimer's estimates, and 60% of them are within 10% of Nettimer's estimates.

In Figure 9, we show the cumulative distribution function of the times spent by SProbe to produce its estimates. SProbe spent at most 400ms to estimate the bottleneck bandwidth of 124 Web servers (i.e., 50% of the measured set). Surprisingly, the largest elapsed time (13.6 seconds) in Figure 9 was spent by SProbe estimating the bandwidth to a server in Salt Lake City. After inspecting the trace, we found that it took 13.4 seconds for this server to send the first data packet after receiving the HTTP GET-request, an unavoidable delay for any network measurement tool.

C. Evaluation of the Heuristic Tests

During the course of gathering the large-scale network measurements reported in Section V-A, we recorded all instances that triggered the cross-traffic detection heuristics. Table II presents the proclivity of these heuristics to filter trials. From this, we see that a large fraction of the trials (63%) were filtered by the heuristics.

An interesting artifact of how the measurement project was conducted is that the bandwidth to some of the peers was reestimated up to five times, allowing SProbe to capture a "clean" measurement, i.e., one without cross traffic. This gives us an opportunity to evaluate the circumstances in which some of the measurements were discarded by the consistent arrival heuristic test. In particular, for peers that had both clean and failed trials, we compared the estimates produced by the failed trials to those produced by the clean trials. Note that because of the nature of the shuffled arrival heuristic test, a trial which fails this test cannot have an estimate associated with it, since packet pairs are no longer consecutively received. As a result, we do not perform a similar comparison for the shuffled heuristic.

Figure 10 presents the cumulative distribution of the ratio between failed and clean consistent arrival trials. A bias exists: Figure 10 shows only the peers to which SProbe was able to make a clean estimate, whereas peers whose estimates never pass this heuristic test are not present (as there are no clean measurements to compare). Although we expected this heuristic test to eliminate some of the estimates where cross-traffic was present, in practice, 95% of the time these estimates are within a factor of two of

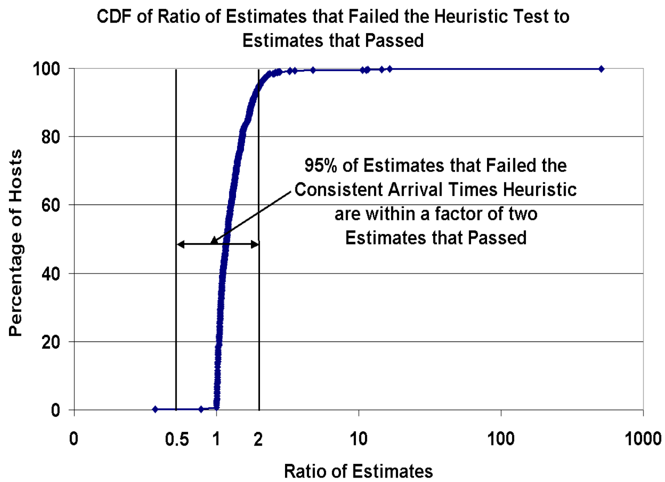

Fig. 10. **Constant Arrival Times Heuristic Test: CDF of the ratio of failed trials to the trials that passed the test.**

the clean estimates. Nevertheless, this ratio is greater that one for almost all of the trials; in other words, cross traffic tends to lead to overestimates of bottleneck bandwidth. A possible explanation is that the heuristic fires specifically in those cases in which cross traffic is interposed between train packets, resulting in compression of the packet pair itself. Currently, we have no explanation for why trials that failed the heuristic test tend to be accurate in practice. We are currently investigating this unexpected behavior.

D. Summary of Evaluations

Our evaluations show that SProbe is *accurate*, *fast*, *scalable* and works in *uncooperative environments*. It is precisely these properties that made SProbe viable, useful in large-scale network measurements and in different realistic scenarios. A tradeoff between speed and accuracy exists. Our evaluations, however, show that SProbe is fast without a significant degradation in accuracy.

VI. Reflections on the Future of Network Measurements

The code for SProbe is freely available and can be downloaded from our website[5]. Currently, SProbe can be run on both Linux and FreeBSD systems and supports the HTTP and the Gnutella application level protocols.

Although, SProbe's behavior is benign, several of the hosts captured by the large-scale measurements perceived SProbe as hostile and intrusive. We received e-mails identifying our measurement host as the originator of break-in attempts. Most of the these e-mails seemed to be sent be home users, rather than professional system administrators. We believe that this is a result of the recent proliferation of firewalls to home users, and the inability of typical home users to distinguish hostile from benign behavior.

Our experience with SProbe and other measurement tools demonstrate that the Internet has migrated from a *friendly* environment (i.e., one in which an implicit assumption of good behavior exists among hosts), to a hostile environment which tends to associate unfamiliar communication patterns with a break-in attempt. Therefore, we expect that building measurement tools will become an increasingly daunting task in the future. Since we anticipate an increase in the implicit level of hostility on the Internet, support for cooperative measurement environments and cooperative tools must be added into future network infrastructure and network protocols.

Finally, we are delighted to report that based on collaboration with the authors of Nettimer, SProbe's techniques of estimating downstream bottleneck bandwidth have been incorporated in the latest version of their tool. This further demonstrates the utility and practicality of SProbe's technique to the context of wide-area, Internet measurements.

VII. Conclusions and Future Work

The main contribution of this paper is the design, implementation, and evaluation of SProbe, a bottleneck bandwidth estimation tool that is *fast*, *accurate*, *scalable* and works in *uncooperative environments*. Furthermore, we present a qualitative comparison of the strengths and weakness of existing bottleneck bandwidth measurement tools and techniques. By examining the requirements of different applications for bandwidth estimation tools (a large-scale network measurement project, building more robust peer-to-peer and multicast overlays, and using bottleneck bandwidth as a factor of decision in dynamic server selection or placement), we conclude that no current tool offers the right set of properties to be useful in practice.

We then report on our experiences using SProbe in a large-scale network measurement project. From these experiences, we quantitatively demonstrate SProbe's accuracy, and also show that its ability to generate fast estimates does not degrade its accuracy. Finally, we make recommendations to future protocol and network designers.

In the future, we plan on studying different heuristic techniques for detecting cross traffic while maintaining the speed of the tool. More specifically, we hope to experiment with the IP identifier field to detect cross traffic. We also plan on adding similar heuristic techniques of estimating bandwidth in the upstream direction. In addition, we are studying ways of incorporating SProbe's techniques into the design of several wide-area systems that would benefit from rapid, accurate bandwidth estimation.

[5] `http://www.cs.washington.edu/homes/tzoompy/sprobe`

ACKNOWLEDGMENTS

We would like to thank Mary Baker, David Ely, Mike Jones, Kevin Lai, Stefan Savage, Neil Spring, Geoff Voelker, David Wetherall, and Andrew Whitaker for their helpful comments on this research.

REFERENCES

[1] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Content-Addressable Network," in *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.

[2] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.

[3] B.Y. Zhao, K.D. Kubiatowicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing," Tech. Rep. UCB//CSD-01-1141, University of California at Berkeley Technical Report, April 2001.

[4] Peter Druschel and Antony Rowstron, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," in *Proceedings of the Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schoss Elmau, Germany, May 2001.

[5] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," in *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, August 2001.

[6] Mark E. Crovella and Robert L. Carter, "Dynamic Server Selection in the Internet," in *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, August 1995.

[7] Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang, "On the Placement of Internet Instrumentation," in *Proceedings of the IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[8] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker, "On the Placement of Web Server Replicas," in *Proceedings of the IEEE INFOCOM 2001*, Anchorage, AK, USA, April 2001.

[9] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson, "TCP Congestion Control with a Misbehaving Receiver," in *ACM Computer Communications Review*, October 1999, vol. 29, pp. 71–78.

[10] Kevin Lai, "Nettimer," `http://mosquitonet.stanford.edu/~laik/projects/nettimer/`, 2000.

[11] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of the ACM SIGCOMM '88 Conference*, Stanford, CA, USA, August 1988.

[12] Srinivasan Keshav, "A Control-Theoretic Approach to Flow Control," in *Proceedings of the ACM SIGCOMM 1991 Technical Conference*, September 1991.

[13] Jean-Chrysostome Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," in *Proceedings of the ACM SIGCOMM 1993 Technical Conference*, August 1993, pp. 289–298.

[14] Robert L. Carter and Mark E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," Tech. Rep. BU-CS-96-006, Computer Science Department, Boston University, March 1996.

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

[16] Kevin Lai and Mary Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay," in *Proceedings of the ACM SIGCOMM 2000 Technical Conference*, Stockholm, Sweden, August 2000.

[17] Constantinos Dovrolis, Parmesh Ramanathan, and David Moore, "What Do Packet Dispersion Techniques Measure?," in *Proceedings of the IEEE INFOCOM 2001*, Anchorage, AK, USA, April 2001.

[18] Van Jacobson, "Pathchar," August 1997, `http://www.caida.org/tools/utilities/others/pathchar/`.

[19] Bruce A. Mah, "pchar," 1999, `http://www.caida.org/tools/utilities/others/pathchar/`.

[20] Allen B. Downey, "Clink," 1999, `http://rocky.wellesley.edu/downey/clink/`.

[21] Robert Carter, "Cprobe and bprobe tools," `http://cs-people.bu.edu/carter/tools/Tools.html`, 1996.

[22] Constantinos Dovrolis, "pathrate," `http://www.cis.udel.edu/~dovrolis/bwmeter.html`, 2001.

[23] Steven M. Bellovin, "A Best-Case Network Performance Model," February 1992, `http://www.research.att.com/~smb/papers/netmeas.pdf`.

[24] Allen B. Downey, "Using pathchar to estimate Internet link characteristics," in *Proceedings of the Special Interest Group on Data Communication (SIGCOMM)*, Cambridge, MA, USA, September 1999.

[25] Kevin Lai and Mary Baker, "Measuring Bandwidth," in *Proceedings of the IEEE INFOCOM 1999*, New York, NY, USA, March 1999.

[26] Kevin Lai and Mary Baker, "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[27] Napster, ," May 1999, `http://www.napster.com`.

[28] Clip2, "The Gnutella Protocol Specification v.0.4," March 2001, `http://www.clip2.com/GnutellaProtocol04.pdf`.

[29] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," Submitted to Multimedia Computing and Networking Conference, January 2002.

[30] Stefan Savage, "Sting: a TCP-based Network Measurement Tool," in *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, Boulder, CO, USA, October 1999, pp. 71–79.

[31] Jitendra Padhye and Sally Floyd, "Identifying the TCP Behavior of Web Servers," in *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, August 2001.

[32] Tim Bray, "Measuring the Web," in *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996, pp. 993–1005.

[33] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul, "Rate of Change and other Metrics: a Live Study of the WWW," in *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, Monterey, CA, USA, December 1997, pp. 147–158.

[34] Ahsan Habib and Marc Abrams, "Analysis of Bottlenecks in International Internet Links," in *Proceedings of the International Conference of Computers and Information Technology (ICCT)*, Dhaka, Bangladesh, 1998.

[35] Vern Paxson and Sally Floyd, "Why we don't know how to simulate the internet," in *Proceedings of the 1997 Winter Simulation Conference*, 1997.

[36] Clip2, "Gnutella measurement project," May 2001, `http://www.clip2.com/`.