Appeared in Proceedings of 1st Usenix Symposium on Internet Technologies and Systems (USITS ’97) Monterey, CA, December 1997.

SPAND: Shared Passive Network Performance Discovery

Srinivasan Seshan
srini@watson.ibm.com

IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

Mark Stemm, Randy H. Katz
{stemm,randy}@cs.berkeley.edu

Computer Science Division
University of California at Berkeley
Berkeley, CA 94720

Abstract

In the Internet today, users and applications must often make decisions based on the performance they expect to receive from other Internet hosts. For example, users can often view many Web pages in low-bandwidth or high-bandwidth versions, while other pages present users with long lists of mirror sites to chose from. Current techniques to perform these decisions are often ad hoc or poorly designed. The most common solution used today is to require the user to manually make decisions based on their own experience and whatever information is provided by the application. Previous efforts to automate this decision-making process have relied on *isolated, active* network probes from a host. Unfortunately, this method of making measurements has several problems. Active probing introduces unnecessary network traffic that can quickly become a significant part of the total traffic handled by busy Web servers. Probing from a single host results in less accurate information and more redundant network probes than a system that shares information with nearby hosts. In this paper, we propose a system called SPAND (**S**hared **Pa**ssive **N**etwork Performance **D**iscovery) that determines network characteristics by making *shared, passive* measurements from a collection of hosts. In this paper, we show why using passive measurements from a collection of hosts has advantages over using active measurements from a single host. We also show that sharing measurements can significantly increase the accuracy and timeliness of predictions. In addition, we present a initial prototype design of SPAND, the current implementation status of our system, and initial performance results that show the potential benefits of SPAND.

1. Introduction

In today’s Internet, it is impossible to determine in advance what the network performance (e.g., available bandwidth, latency, and packet loss probability) between a pair of Internet hosts will be. This capability is missing in today’s suite of Internet services, and many applications could benefit from such a service:

- Applications that are presented with a choice of hosts that replicate the same service. Specific examples of this are FTP and Web mirror sites and Harvest caches that contact the “closest” peer cache. Today, these applications rely on statistics such as hop count/routing metrics [9], round-trip latency [7], or geographic location [10]. However, each of these techniques has significant weaknesses, as we show in Section 2.
- Web clients that have a choice of content *fidelity* to download from a Web server, e.g., a full graphics representation for high-bandwidth connectivity or a text-only representation for low-bandwidth connectivity. Today, the user must manually select the fidelity of the content that they wish to view, sometimes making overaggressive decisions such as viewing no images at all.
- Applications that provide feedback to the user that indicates the expected performance to a distant site. For example, Web browsers could insert an informative icon next to a hyperlink indicating the expected available bandwidth to the remote site referred to by the hyperlink.

Each of these applications needs the ability to determine in advance the expected network performance between a pair of Internet hosts. Previous work in this area has relied on *isolated, active* network probing from a single host to determine network performance characteristics. There are two major problems with this approach:

- Active probing requires the introduction of unnecessary traffic into the network. Clearly, an approach that determines the same information with a minimum of unnecessary traffic is more desirable. We also show later that this unnecessary traffic can quickly grow to become a non-negligible part of the traffic reaching busy Web servers, reducing their efficiency and sometimes their scalability.

- Probing from a single host prevents a client from using the past information of nearby clients to predict future performance. Recent studies [2][20] have shown that network performance from a client to a server is often stable for many minutes and very similar to the performance observed by other nearby clients, so there are potential benefits of sharing information between hosts. In Section 3.2, we show examples where using shared rather than isolated information increases the likelihood that previously collected network characteristics are valid.

We are developing a system called SPAND (**S**hared **Pa**ssive **N**etwork Performance **D**iscovery) that overcomes the above problems of isolated active probing by collecting network performance information *passively* from a collection of hosts, *caching* it for some time and *sharing* this information between them. This allows a group of hosts to obtain timely and accurate network performance information in a manner that does not introduce unnecessary network traffic.

The rest of this paper is organized as follows. In Section 2, we describe related work in more detail. In Section 3, we point out the advantages and challenges of using passive shared measurements over isolated active measurements. In Section 4, we present a detailed design of SPAND. In Section 5, we describe the implementation status of SPAND and initial performance results, and in Section 6, we conclude and describe future work.

2. Related Work

In this section, we describe in more detail previous work in network probing algorithms and server selection systems.

2.1 Probing Algorithms

A common technique to estimate expected performance is to test the network by introducing probe packets. The objective of these probes is to measure the round trip latency, peak bandwidth or available "fair-share" bandwidth along the path from one host to another

Probes to measure round-trip latency and peak bandwidth are typically done by sending groups of back-to-back packets to a server which echoes them back to the sender. These probes are referred to as NetDyn probes in [4], packet pair in [13], and bprobes in [6]. As pointed out in earlier work on TCP dynamics [12], the spacing between these packets at the bottleneck link is preserved on higher-bandwidth links and can be measured at the sender.

If the routers in the network do not implement fair queuing, the minimum of many such measurements is likely to be close to the raw link bandwidth, as assumed in other work ([4][6][19]). Pathchar [19] combines this technique with traceroute [22] to measure the link bandwidth and latency of each hop along the path from one Internet host to another. Packet Bunch Mode (PBM) [20] extends this technique by analyzing various sized groups of packets inserted into the network back-to-back. This allows PBM to handle multi-channel links (i.e. ISDN connections, muiti-link Point-to-Point Protocol (PPP) links, etc.) as well as improve the accuracy of the resulting measurements.

If routers in the network implement fair queuing, then the bandwidth indicated by the back-to-back packet probes is an accurate estimate of the "fair share" of the bottleneck link's bandwidth [13]. Another fair share bandwidth estimator, cprobe [6], sends a short sequence of echo packets from one host to another as a simulated connection (without minimal flow control and no congestion control). By assuming that "almost-fair" queuing occurs over the short sequence of packets, cprobe provides an estimate for the available bandwidth along the path from one host to another. Combined with information from bprobes, cprobes can estimate the competing traffic along the bottleneck link. However, it is unclear how often this "almost-fair" assumption is correct and how accurate the resulting measurements are. TReno [15] also uses ICMP echo packets as a simulated connection, but uses flow control and congestion control algorithms equivalent to that used by TCP.

The problem with these methods is that they can introduce significant amounts of traffic that is not useful to any application. For example, pathchar sends at least tens of kilobytes of probe traffic per hop, and a cprobe sends 6 kilobytes of traffic per probe. This amount of probe traffic is a significant fraction (approximately 20%) of the mean transfer size for many Web connections ([1], [2]) as well as a significant fraction of the mean transfer size for many Web sessions. We discuss in more detail the limitations of active probing in Section 3.3.

2.2 Server Selection Systems

Many server selection systems use network probing algorithms to identify the closest or best connected server. For example, Carter et al. at Boston University [5] use cprobes and bprobes to classify the connectivity of a group of candidate mirror sites. Harvest [7] uses round-trip latency to identify the best peer cache from which to retrieve a Web page. Requests are initiated to

System	What measured/ used to identify performance	Additional traffic introduced	Notes	Where Deployed
Bprobes, Cprobes	Peak and Available Bottleneck Bandwidth	Significant (~10K)	Cprobes uses no flow/congestion control	Client Side
Packet Pair	Available Bandwidth	Little (~1K)	Assumes per-flow fair queuing	Client Side
Pathchar	Hop-by-hop link bandwidth, latency	Significant (>10K)	No congestion control	Client Side
Packet Bunch Modes	Peak Bottleneck Bandwidth	Significant (~10K)		Client Side
Treno	Available Bandwidth	Significant (>10K)	Uses TCP Flow/Congestion Control	Client Side
IPV6 Anycast	Routing Metric	Little (routing data and queries)		Internal Network
Harvest	Latency	Little (~1K)		Server Side
HOPS	Routing Metric	Little (routing data and queries)		Internal Network
DistributedDirector	Routing Metric	Little (routing data and remote queries)		Server Side and Internal Network
SPAND	**Available Bandwidth, Packet Loss Probability**	**Little (local reports and queries)**		**Client Side**

TABLE 1. Summary of Previous Work compared to SPAND

each peer cache, and the first to begin responding with a positive answer is selected and the other connections are closed. Other proposals [10] rely on geographic location for selecting the best cache location when push-caching Web documents.

There are also preliminary designs for network-based services to aid in server selection. IPV6's Anycast [11][18] service provides a mechanism that directs a client's packets to any one of a number of hosts that represent the same IP addresses. This service uses routing metrics as the criteria for server selection. The Host Proximity Service (HOPS) [9] uses routing metrics such as hop counts to select the closest one of a number of candidate mirror sites.

Cisco's DistributedDirector [8] product relies on measurements from Director Response Protocol (DRP) servers to perform efficient wide area server selection. The DRP servers collect Border Gateway Protocol (BGP) and Interior Gateway Protocol (IGP) routing table metrics between distributed servers and clients. When a client connects to a server, DistributedDirector contacts the DRP server for each replica site to retrieve the information about the distance between the replica site and the client.

The problem with many of these approaches is that one-way latency, geographic location, and hop count are often poor estimates of actual completion time. Other work [5][17] shows that hop count is poorly correlated with available bandwidth, and one-way latency does not take available bandwidth into account at all. Even those systems that provide better performance metrics [5] rely on each end host independently measuring network performance.

Another design choice to consider is where the system must be deployed. A system that is deployed only at the endpoints of the network is easier to maintain and deploy than a system that must be deployed inside the internal infrastructure, and a system that is deployed only at the client side is easier to deploy than a system that relies on client and server side components.

Table 1 summarizes the previous work in this area. The significant shortcomings of existing network performance discovery and server selection systems are:

- Introduction of new traffic into the network that can quickly become significant when compared to "useful" traffic.
- Reliance on measurements from a single host, which are more often redundant and inaccurate than measurements from a collection of hosts.
- Use of metrics such as hop count, latency, and geographic location as imprecise estimates of available bandwidth.

We discuss these shortcomings further in the next section.

3. Passive and Cooperative Measurements

The goal of our work is to provide a unified repository of actual end-to-end performance information for applications to consult when they wish to determine the network performance to distant Internet hosts. Our approach addresses the shortcomings of previous work in 2 ways: (1) relying solely on passive rather than active measurements of the network, and (2) sharing measurement information between all hosts in a local network region. In this section we show the potential benefits and challenges of using shared, passive measurements to predict network performance instead of using isolated, active measurements.

3.1 Network Performance Stability

In order for past transfers observed by hosts in a region to accurately predict future performance, the network performance between hosts must remain relatively stable for periods of time on the order of minutes. Without this predictability, it would be impossible for shared passive measurements of the network to be meaningful. Past work has shown evidence that this is true for some Internet hosts [2][20]. We wanted to verify these results in a different scenario.

To understand more closely the dynamics of network characteristics to a distant host, we performed a controlled set of network measurements. This consisted of repeated HTTP transfers between UC Berkeley and IBM Watson. For a 5 hour daytime period (from 9am PDT to 2pm PDT), a Web client at UC Berkeley periodically downloaded an image object from a Web server running at IBM Watson. Although this measurement is clearly not representative of the variety of connectivity and access patterns that exist between Internet hosts, it allowed us to focus on the short-term changes in network characteristics that could occur between a pair of well-connected Internet hosts separated by a large number of Internet hops.

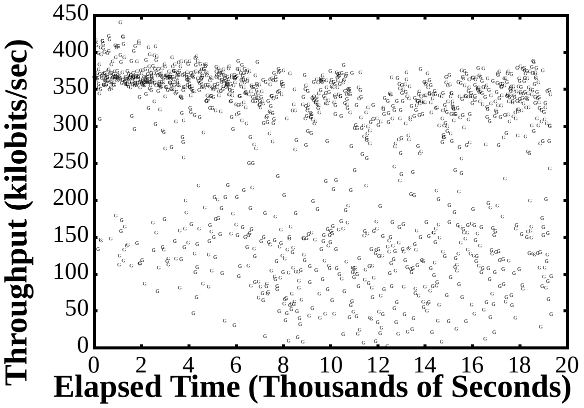

Figure 1. Throughput from UC Berkeley to IBM during a 5 hour daytime period

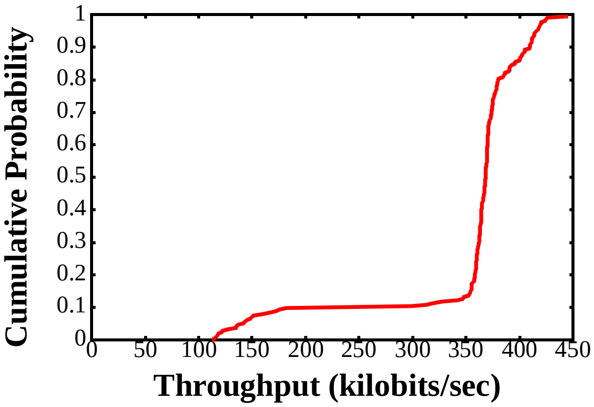

Figure 2. CDF of throughput from UC Berkeley to IBM: initial 30 minutes

Figure 1 shows the raw throughput measurements as a function of time over the 5 hour period. We see that in the first 30 minutes of the trace, one group of measurements is clustered around 350 kilobits/sec (presumably the available bandwidth on the path between UC Berkeley and IBM). A smaller group of measurements has lower throughputs, at 200 kilobits/sec and lower. This second group of connections presumably experiences one or more packet losses. This clustering is more clearly shown in Figure 2, the cumulative distribution function (CDF) of throughputs during the first 30 minutes of the trace. As the day progresses, two things change. The available bandwidth decreases as the day progresses, and a larger fraction of transfers experience one or more packet losses. This effect is shown in Figure 3, the cumulative distribution function of throughputs for the entire 5 hour period. More samples are clustered around lower throughput values and there is more variation in the available bandwidth. However, there is still a noticeable separation between the two

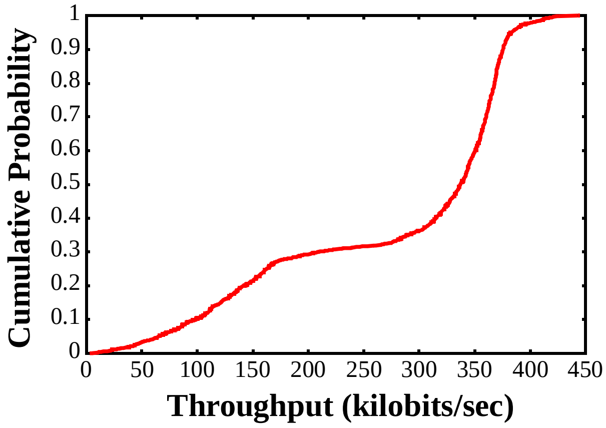

Figure 3. CDF of throughput from UC Berkeley to IBM: full 5 hour period

groups of throughput measurements.

This distribution of performance suggests that although the distribution of throughputs changes as the day progresses, a system like SPAND could still provide meaningful performance predictions. Although the performance distribution of the early part of the day and the later part of the day are quite different, they each have lifetimes of tens of minutes or more. Even when aggregating all the different performance measurements for the entire 5 hour period, approximately 65% of the throughput samples are within a factor of 2 of the median throughput.

3.2 Shared Measurements: Benefits and Challenges

Using shared rather than isolated measurements allows us to eliminate redundant network probes. If two hosts are nearby each other in terms of network topology, it is likely that they share the same bottleneck link to a remote site. As a result, the available bandwidth they measure to this site is likely to be very similar [2]. Therefore, it is unnecessary and undesirable for each of these hosts to independently probe to find this information--they can learn from each other.

To quantify how often this information can be shared between nearby hosts, we examined Internet usage patterns by analyzing client-side Web traces. From these traces, we determined how often a client would need to probe the network to determine the network performance to a particular Web server when shared information was and was not available.

More formally, for a single Web server, we can represent the list of arrival times from a single client (or a shared collection of clients) as a sequence (t_1, t_2,..., t_n). If the difference between t_{i+1} and t_i is extremely small (less than ten seconds), we merge the events together into a single Web browsing "session." Clearly, the first arrival always requires a probe of the network. In addition, if we assume that the time between significant network changes is a fixed value Δ, then if t_{i+1}-t_i>Δ, then the client must make a probe to determine the new network characteristics. If t_{i+1}-t_i<Δ, then no probe is necessary. As mentioned previously [2][20], an appropriate value for Δ is on the order of tens of minutes.

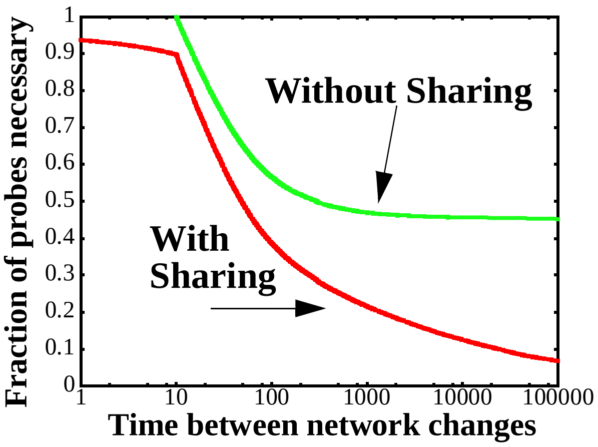

Figure 4. The benefit of sharing. Figure shows the fraction of network probes that are necessary as a function of the time between network state changes

Figure 4 shows the results of this analysis for a particular client-side trace consisting of 404780 connections from approximately 600 users over an 80 hour time period [23]. The x-axis represents the time Δ between network changes, and the y-axis represents the fraction of network probes that are necessary. There are two curves in the figure. The upper curve represents the number of probes that are necessary if no sharing between clients is performed, and the lower curve represents the number of probes that are necessary if clients share information between them. The upper curve begins at Δ=10 seconds because of the "sessionizing" of individual connections described above. We see that the number of probes that are necessary when clients share network information is dramatically reduced. This is evidence that a collection of hosts can eliminate many redundant network probes by sharing information.

The use of shared measurements is not without challenges, however. Measurements from arbitrary hosts in a region cannot be combined. For example, it is necessary to separate modem users within a local domain from LAN users in the same domain, because the two sets of users may not share the same bottleneck link. Similarly, hosts in a local domain may use different TCP implementations that result in widely varying performance. The challenge is that it is often difficult to determine who the set of "similarly connected" hosts within an local domain are. We can use the topology of the local domain along with post-processing on past measurements to determine which network subnets exhibit significantly different performance. The system can then coalesce these subnets together into classes of equivalent connectivity and avoid aggregating measurements from dissimilar hosts.

3.3 Passive Measurements: Benefits and Challenges

The use of passive measurements avoids the introduction of useless probe traffic into the network. This advantage over active probing systems comes at the expense of making the job of measuring available bandwidth more difficult. However, this advantage is critical since probe traffic can sometimes become a measurable fraction of the traffic handled by busy Web servers. For example, consider the scenario of mirror sites that replicate the same content. In an active probing system, a client must first contact each of the mirror sites to determine which mirror is the "best." This slows down servers with probe-only traffic and limits the scalability of such a system.

The following thought experiment shows why. Consider a Web server with a variable number of mirror sites. Assume that each mirror site is connected to the Internet via a 45 Mbit/second T3 link and assume that the mean transfer size is 100 kbytes and the mean probe size is 6 kbytes. These are optimistic estimates; most Web transfers are shorter than 100 kbytes and many of the network probing algorithms discussed in Section 2.1 introduce more than 6 kbytes. From a network perspective, an estimate of the number of requests per second that the collection of mirrors can support is the aggregate bandwidth of the mirrors' Internet links divided by the sum of the average Web transfer size and any associated probe traffic for the transfer. Figure 5 shows the number of requests per second that such a system can support as a function of the number of mirror sites for two systems: one without probe traffic, and one with probe traffic. We see that the system without probe traffic scales perfectly with the number of mirrors. For the system with probe traffic, however, for each Web request that is handled by a single mirror, a network probe must be sent to all of the other mirrors. On the server side, this means that for each Web request a particular mirror site handles, it must also handle a probe request from clients being serviced at every other mirror location. As the number of mirrors increases, the number of requests served per second becomes limited by the additional probe traffic.

There are challenges in using passive network measurements, however. Using passive rather than active measurements is difficult for several reasons:

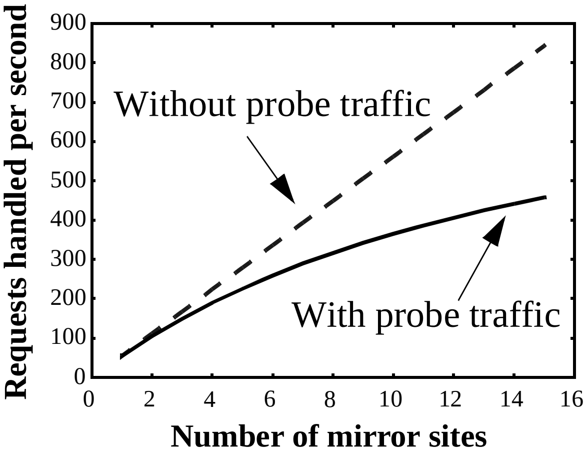

Figure 5. The effect of probe traffic on scalability. Figure shows requests/second that mirrors can serve as a function of the number of mirror sites

- Passive measurements are uncontrolled experiments, and it can be difficult to separate network events from those occurring at the endpoints, such as a rate-limited transmission or a slow or unreachable server.
- Passive measurements are only collected when a host contacts a remote site. In order to have timely measurements, hosts in a local domain must visit distant hosts often enough to obtain timely information. If this is not true, the client may obtain either out-of-date information or no information at all.

For our purposes, there is no need to distinguish between network events and endpoint events. If a remote site is unreachable or has poor connectivity because it is down or overloaded, that information is just as useful. It is important to distinguish between rate-controlled and bulk transfer connections so the performance numbers from one are not used to estimate performance for the other. We can distinguish between rate-controlled and bulk transfer transmissions by using TCP and UDP port numbers and the application classes described in Section 4.

To identify if passive measurements can provide timely information, we must analyze typical Internet usage patterns and determine how often passive techniques lead to out-of-date information. We can use the results shown in Figure 4 to see this. We can model the arrival pattern of clients as a sequence of times ($t_1 ... t_n$) as before. In the passive case, when $t_{i+1}-t_i>\Delta$, instead of saying that an active probe is necessary, we say that the passively collected information has become out of date. So the fraction of time that an active probe is necessary is exactly the same as the fraction of time that passive measure-

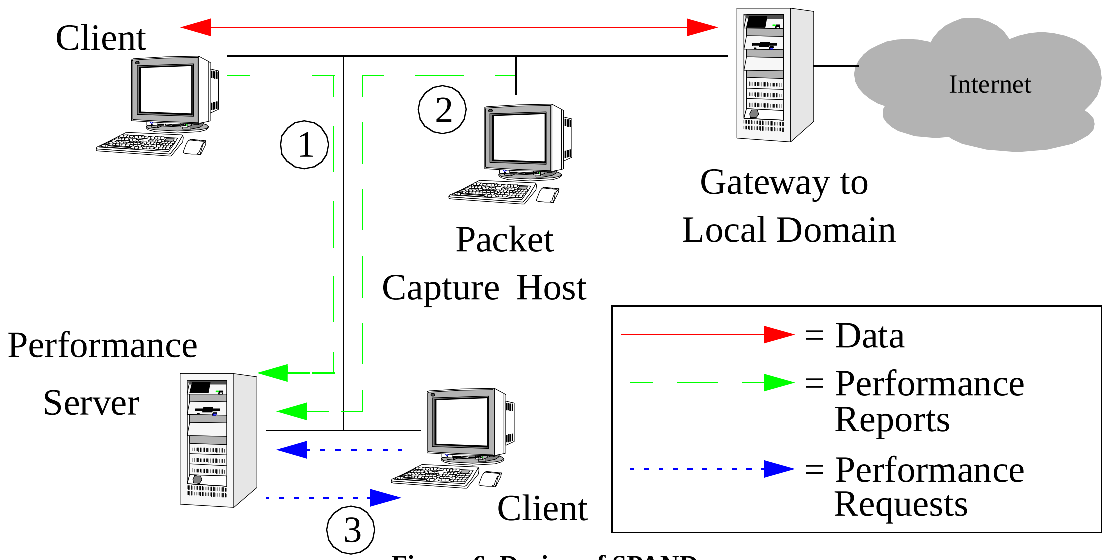

Figure 6. Design of SPAND

ments become out of date. As mentioned earlier, the appropriate value for Δ is on the order of tens of minutes. We see that even a relatively small collection of hosts can obtain timely network information when sharing information between them. If we assume that network conditions change approximately every 15 minutes, then the passive measurements collected from this relatively small collection of 600 hosts will be accurate approximately 78% of the time. For larger collections of hosts (such as domain-wide passive measurements), the availability of timely information will be even greater, as shown in Section 5.3.

4. Design of the SPAND System

In this section, we describe the design for SPAND, including steps for incremental deployment in existing networks.

Figure 6 shows a diagram of the components of SPAND. SPAND is comprised of *Clients*, *Performance Servers*, *and Packet Capture Hosts*. Clients have modified network stacks that transmit *Performance Reports* to Performance Servers. These reports contain information about the performance of the network path between the client and distant hosts. The format of a Performance Report is shown in Figure 7, and includes parameters such as connection bandwidth and packet loss statistics. The Transport Protocol field indicates the type of transport connection (UDP or TCP) used by the initiator of the connection. The optional Application Class field is a hint as to the way in which the application is using the transport connection. If an Application Class is not provided, the Performance Server can use the Port Number and Transport Protocol fields to make a guess for the application class.

0	15	16	31
Version	Type	Transport Pr.	App.Class
Source IP Address			
Source Port		Dest Port	
Dest IP Address			
NTP Timestamp, most sig word			
NTP Timestamp, least sig word			
Length of Sample in octets			
Duration of Sample in ms			
Total Packets Received			
Total Packets Lost			
Packet Size in octets			

Figure 7. Format of a Performance Report

The Application Class field is desirable because not all applications use transport connections in the same way. Some applications (such as Web browsers and FTP programs) use TCP connections for bulk transfers and depend on the reliability, flow control, and congestion control abstractions that TCP connections provide. Applications such as telnet primarily use TCP connections for reliability and not for flow or congestion control. Other applications such as RealAudio in TCP mode use TCP connections for different reasons such as the ability to traverse firewalls. The transport level network performance reported from different applications may

0	15	16	31
Version	Type	Protocol	App. Class
Request IP Address			

Figure 8. Format of a Performance Request

0	15	16	31
Version	Type	Protocol	App. Class
Response IP Address			
Expected Available Bandwidth (kbits/sec)			
Std Dev of Available Bandwidth (kbits/sec)			
Expected Packet Loss Probability			
Std Dev of Expected Loss Probability			

Figure 9. Format of a Performance Response

vary widely depending on the way the transport connection is used, and we want to separate these performance reports into distinct classes.

In addition, many applications may intermittently change the way in which a connection is used. For example, a passive FTP connection may switch from transporting control information to transferring bulk data. Similarly, a persistent HTTP 1.1 connection may have idle “think” periods where the user is looking at a Web page as well as bulk transfer periods. To properly account for all these variations, we need applications to take part in the performance reporting process. Our toolkit provides an API which enables applications to start a measurement period on a connection as well as end the measurement and automatically send a report to the local Performance Server.

A Performance Server receives reports from all clients in a local area and must incorporate them into its performance estimates. The server maintains different estimates for different classes of applications as well as different classes of connectivity within its domain. In addition, the Performance Server can also identify reports that have possibly inaccurate information and discard them. Clients may later query the information in the server by sending it a Performance Request containing an (Address, Application Class) pair. The server responds to it by returning a Performance Response for that pair, if one exists. This response includes the Performance Server’s estimates of available bandwidth and packet loss probability from the local domain to the specified foreign host. The request and response formats are shown in Figures 8 and 9.

4.1 Mechanisms for Incremental Deployment

The system described in the previous section is an ideal endpoint we would like to reach. In practice, it may be difficult to immediately modify all client applications to generate performance reports, especially since many clients may need modifications to their protocol stacks to make the statistics necessary for SPAND available. To quickly capture performance from a large number of end clients, a Packet Capture Host can be deployed that uses a tool such as BPF [16] to observe all transfers to and from the clients. The Packet Capture Host determines the network performance from its observations and sends reports to the Performance Server on behalf of the clients. This allows a large number of Performance Reports to be collected while end clients are slowly upgraded. The weakness of this approach is that a number of heuristics must be employed to recreate application-level information that is available at the end client. Section 5.2 describes these heuristics in more detail.

4.2 Example Scenario

This example scenario using Figure 6 illustrates the way in which the agents that make up SPAND coordinate. Assume that a user is browsing the Web. As the user is browsing, the user’s application generates Performance Reports and sends them to the local Performance Server (1 in the figure). Also, a Packet Capture Host deployed at the gateway from the local domain to the Internet generates Performance Reports on behalf of the hosts in the domain (2 in the figure). Later, some other user reaches a page where she must select between mirror locations. The Web browser makes a Performance Query to the local Performance Server and gets a response (3 in the figure). The Web browser uses the Performance Response to automatically contact the mirror site with the best connectivity.

5. Implementation Status and Performance

In this section, we describe the current implementation status of SPAND and present initial performance measurements from a working SPAND prototype.

5.1 Implementation Details

SPAND is organized as a C++ toolkit that provides object abstractions for the agents described above. Application writers can create objects for agents such as PerformanceReporter(), PerformanceReportee(), PerformanceRequestor(), and PerformanceRequestee() and use these objects to make, send or receive reports, or make queries about network performance. We also have partial implementations of the toolkit in Java and Perl.

Using the SPAND toolkit, we have implemented the

Packet Capture Host, Performance Server and a simple text-based SPAND Client. We have also written several client applications that use the SPAND toolkit to make use of Performance Reports. We have written a HTTP proxy using the Perl libwww [14] library and the SPAND toolkit that modifies HTML pages to include informative icons that indicate the network performance to a distant site mentioned in a hyperlink. This is not the first application of this type; others have been developed at IBM [3] and at Boston University [5]. However, our application uses actual observed network performance from local hosts to make decisions about the icon to insert in the HTML page.

We have also written a Java-based application that allows the user to obtain an overview of the connectivity from a local domain to distant hosts. This application shows the number of performance reports collected for all hosts as well as the details about the reported network statistics for a given host. This tool was used to generate the graphs in Section 3.1.

5.2 Packet Capture Host Policies

Because our Packet Capture Host is not located at end clients, it does not have perfect information about the way in which applications use TCP connections. This can lead to inaccurate measurements of network characteristics such as bandwidth. For example, if a Web browser uses persistent or keep-alive connections to make many HTTP requests over a single TCP connection, then simply measuring the observed bandwidth over the TCP connection will include the gaps between HTTP requests in the total time of the connection, leading to a reduction in reported bandwidth. To account for this effect, we modified the Packet Capture Host to use heuristics to detect these idle periods in connections. When the Packet Capture Host detects an idle period, it makes two reports: one for the part of the connection before the idle period, and another for the part of the connection after the idle period. The ratio measurements in Section 5.3 include systems with and without the use of these heuristics. Another example is when Common Gateway Interface (CGI) programs are executed as part of HTTP requests. The idle time when the server is executing the CGI program leads to an artificially low reported bandwidth and does not reflect the performance of other HTTP requests to the same server. Ideally, the Packet Capture Host would treat these connections as a different Application class. For the purposes of these measurements, however, the Packet Capture Host excluded the idle periods and generated multiple Performance Reports as above.

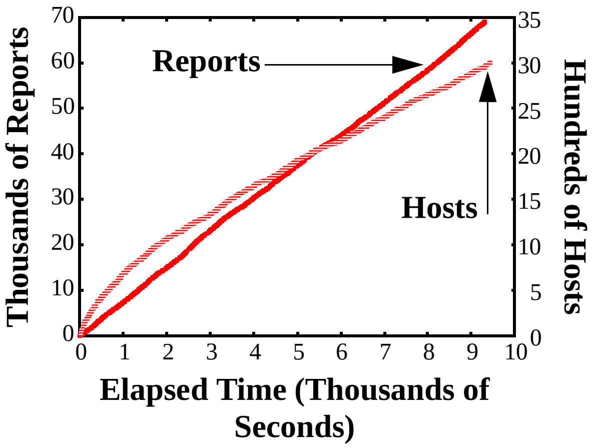

Figure 10. Cumulative number of reports generated and hosts reported about as a function of time

5.3 Performance

There are several important metrics by which we can measure the performance of the SPAND system:

1. How long does it take before the system can service the bulk of Performance Requests?
2. In the steady-state, what percentage of Performance Requests does the system service?
3. How accurate are the performance predictions?

To test the performance of our system, we deployed a Packet Capture Host at the connection between IBM Research and its Internet service provider. Hosts within IBM communicate with the Internet through the use of a SOCKSv4 firewall [21]. This firewall forces all internal hosts to use TCP and to initiate transfers (i.e. servers can not be inside the firewall). The packet capture host monitored all traffic between the SOCKS firewall at IBM Research and servers outside IBM's internal domain. The measurements we present here are from a 3 hour long weekday period. During this period, 62,781 performance reports were generated by the packet capture host for 3,008 external hosts. At the end of this period, the Performance Server maintained a database of approximately 60 megabytes of Performance Reports. Figure 10 shows the cumulative number of reports generated and hosts reported about as a function of time.We see that about 10 reports are generated per second, which results in a network overhead of approximately 5 kilobits per second. We also see that while initially a large number of reports are about a relatively small number of hosts (the upward curve and leveling off of the curve), as time progresses, a significant number of new hosts are reported about as time progresses. This indicates that the

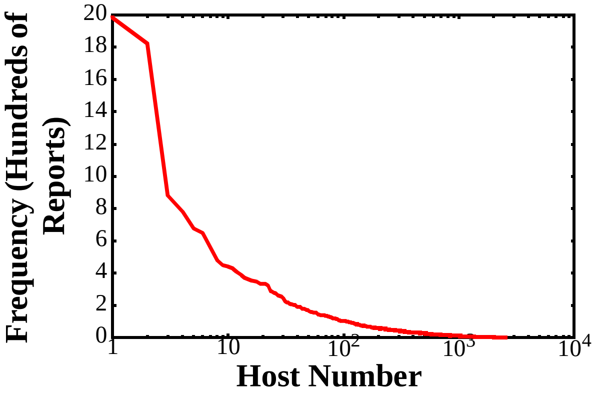

Figure 11. Histogram of number of Performance Reports received per host. The x axis is on a log scale.

"working set" of hosts includes a set of hosts who are reported about a small number of times. This finding is reinforced in Figure 11, which shows a histogram of the number of reports received for each host over the trace. We see that a large majority of hosts receive only a few reports, while a small fraction of hosts receive most of the reports. The mean number of reports received per host was 23.67 and the median number of reports received per host was 7.

To test the accuracy of the system, we had to generate a sequence of Performance Reports and Performance Requests to test the system. Since there are no applications running at IBM that currently use the SPAND system, we assumed that each client host would make a single Performance Request to the Performance Server for a distant host before connecting to that host, and a single Performance Report to the Performance Server after completing a connection. In actual practice, applications using SPAND would probably request the performance for many hosts and then make a connection to only one of them. The performance of the SPAND system on this workload is summarized in Figures 12 and 13.

When a Performance Server is first started, it has no information about prior network performance and cannot respond to many of the requests made to it. As the server begins to receive performance reports, it is able to respond to a greater percentage of requests. Determining the exact "warmup" time before the Performance Server can service most requests is important. Figure 12 shows the probability that a Performance Request can be serviced by the Performance Server as a function of the number of reports since the "cold start" time. We say that a request can be serviced if there is at least one previously collected Performance Report for that host in the Performance Server's database. As we can see from the graph, the Performance Server is able to service 70% of the requests within the first 300 reports (less than 1 minute), and the Performance Server reaches a steady-state service rate of 95% at around 10,000 reports (approximately 20 minutes). This indicates that when a Performance Server is first brought up, there is enough locality in client access patterns that it can quickly service the bulk of the Performance Requests sent to it.

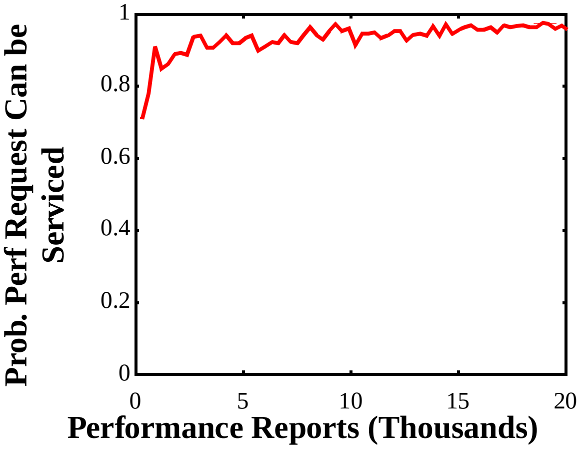

Figure 12. Probability that a Performance Request can be Serviced as a function of the number of Performance Reports.

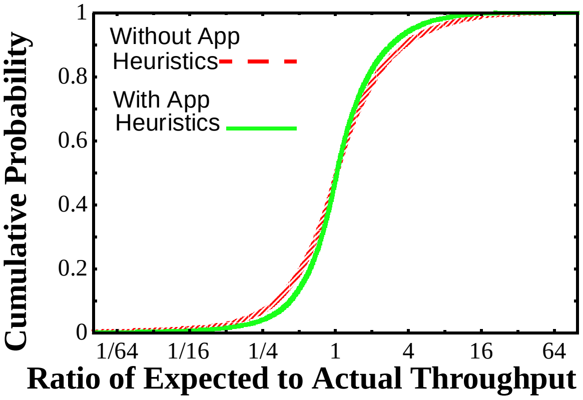

Figure 13. CDF of ratio of expected throughput (as generated by the Performance Server) to actual throughput (as reported by the client). The x axis is on a log scale

To measure the accuracy of Performance Responses, for each connection we computed the ratio of the throughput returned by the Performance Server for that connection's host with the throughput actually reported by the Packet Capture Host for that connection. Figure 13 plots the cumulative distribution function of these ratios. The x axis is plotted on a log scale to equally show ratios that are less than and greater than one. Table 2 shows the

System	% within factor of 2	% within factor of 4
Base System	**59.08%**	**84.05%**
Base System + App Heuristics	**68.84%**	**90.18%**

TABLE 2. Accuracy of Performance Responses

probability that a Performance Response is within a factor of 2 and 4 of the actual observed throughput. We see that Performance Responses are often close to the actual observed throughput. Obviously, different applications will have different requirements as to the error that they can tolerate. Factors of 2 and 4 are shown only as representative data points.

Closer examination of transfers that result in very large or small ratios indicate a few important sources of error in the predictions. Some servers have a mixture of small and large transfers made to them. As a result of TCP protocol behavior, the larger transfers tend to be limited by the available bandwidth to the server whereas the smaller transfers tend to limited by the round trip time to the server. We plan to incorporate round trip measurements into a future version of the SPAND Performance Server to address this problem. In addition, the heuristics used to recreate end client information are effective but not always accurate. Many poor estimates are a result of pauses in the application protocol. This source of inaccuracy will vanish as more end clients participate in the SPAND system and less reliance is made on the Packet Capture Host's heuristics.

6. Conclusions and Future Work

There are many classes of Internet applications that need the ability to predict in advance the network performance between a pair of Internet hosts. Previous work providing this information has depended on isolated, active measurements from a single host. This does not scale to many users and does not provide the most accurate and timely information possible. In this paper, we have proposed a system called SPAND (Shared Passive Network Performance Discovery) that uses passive measurements from a collection of hosts to determine wide-area network characteristics. We have justified the design decisions behind SPAND and presented a detailed design of SPAND and mechanisms for incremental deployment.

Initial measurements of a SPAND prototype show that it can quickly provide performance estimates for approximately 95% of transfers from a site. These measurements also show that 69% of these estimates are within a factor of 2 of the actual performance and 90% are within a factor of 4.

We believe that a number of techniques will improve the accuracy of SPAND's performance estimates.

1. As clients are modified to transmit their own Performance Reports, the accuracy of the reports will improve. This will in turn improve the quality of the estimates that the Performance Server provides.
2. The Performance Server currently returns the median of all past measurements as an estimate of future performance. The measurements presented in this paper were made over a relatively short time scale. As shown in Section 3.1, the distribution of network performance changes as time passes. To provide better estimates, the Performance Server must give newer Performance Reports greater importance and discard information from older reports.
3. The performance of many transfers is limited by the round trip time to the server instead of the available bandwidth. We can improve the quality of SPAND's performance estimates by providing round trip estimates as part of the service and using both throughput and round trip times to predict the duration of a transfer.
4. The Performance Server currently combines the reports of all clients within its domain. It makes no attempt to eliminate poorly configured or misbehaving hosts. Preventing these hosts from impacting the estimates of network performance should reduce persistent sources of error.
5. The Performance Server estimates performance to all IP addresses outside its domain independently. However, network performance to many remote hosts is identical since communications to these hosts share the same bottlenecks. For example, the performance of most connections between the United States and Europe is probably limited by a shared transatlantic bottleneck link. We plan to add *Aggregation Experiments* to the Performance Server that allow it to analyze the distribution of reports to remote hosts over time and combine distant hosts into classes of equivalent connectivity. The server can then use reports from one of the hosts in a class to make or update estimates about the connectivity of other hosts in the same class.

6. Currently, the Packet Capture Host only generates Performance Reports for the bulk transfer Application Class. We plan to modify the Packet Capture Host to generate Performance Reports for other Application Classes such as telnet and server program execution (CGI programs).

7. Acknowledgments

Thanks to Vincent Cina and Nick Trio for providing the access necessary to install SPAND at IBM. Hari Balakrishnan, Steve Gribble, Todd Hodes, and Venkata Padmanabhan provided many useful comments on earlier versions of this paper that helped improve the presentation. This work is supported by DARPA contract DAAB07-95-C-D154 and grants from the California MICRO Program, Hughes Aircraft Corporation, Ericcson, and IBM. Mark is supported by an IBM fellowship.

8. References

[1] M Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proc. ACM SIGMETRICS '96*, May 1996.

[2] H. Balakrishnan, S. Seshan, M. Stemm, and R.H. Katz. Analyzing Stability in Wide-Area Network Performance. In *Proc. ACM SIGMETRICS '97*, June 1997.

[3] R. Barrett, P. Maglio, and D. Kellem. How to Personalize the Web. In *Proc. CHI '97*, 1997.

[4] J.C Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. In *Proc. ACM SIGCOMM '93*, San Francisco, CA, Sept 1993.

[5] R. L. Carter and M. E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report BU-CS-96-007, Computer Science Department, Boston University, March 1996.

[6] R. L. Carter and M. E. Crovella. Measuring bottleneck-link speed in packet switched networks. Technical Report BU-CS-96-006, Computer Science Department, Boston University, March 1996.

[7] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings 1996 USENIX Symposium*, San Diego, CA, Jan 1996.

[8] Cisco Distributed Director Web Page. http://www.cisco.com/warp/public/751/distdir/index.html, 1997.

[9] P. Francis. *http://www.ingrid.org/hops/wp.html*, 1997.

[10] J. Gwertzman and M. Seltzer. The Case for Geographical Push-Caching. In *Proc. Fifth IEEE Workshop on Hot Topics in Operating Systems*, May 1995.

[11] R. Hinden and S. Deering. *IP Version 6 Addressing Architecture*. RFC, Dec 1995. RFC-1884.

[12] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM 88*, August 1988.

[13] S. Keshav. Packet-Pair Flow Control. *IEEE/ACM Transactions on Networking*, February 1995.

[14] libwww-perl-5 home page. http://www.linpro.bo/lwp, 1997.

[15] M. Mathis and J. Mahdavi. Diagnosing Internet Congestion with a Transport Layer Performance Tool . In *Proc. INET '96*, Montreal, Canada, June 1996.

[16] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter '93 USENIX Conference*, San Diego, CA, January 1993.

[17] J. C. Mogul. Network Behavior of a Busy Web Server and its Clients. Technical Report 95/5, Digital Western Research Lab, October 1995.

[18] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*. RFC, Nov 1993. RFC-1546.

[19] pathchar – A Tool to Infer Characteristics of Internet Paths. ftp://ee.lbl.gov/pathchar, 1997.

[20] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, U. C. Berkeley, May 1996.

[21] Socks Home Page. http://www.socks.nec.com, 1997.

[22] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[23] UC Berkeley Annex WWW Traces. http://www.cs.berkeley.edu/ gribble/traces/index.html, 1997.

Paper Title: SPAND: Shared Passive Network Performance Discovery

Authors: Srinivasan Seshan, Mark Stemm (student), Randy H. Katz

Contact: Srinivasan Seshan

IBM T.J. Watson Research Center
P.O Box 704, H3-D27
Yorktown Heights, NY 10598

(914) 784 - 6023 (daytime)
(914) 345 - 9894 (evening)
(914) 784 - 6205 (FAX)
srini@watson.ibm.com