Measuring User-Perceived Internet Performance in Multiple Locations

A Thesis
Presented to
The Academic Faculty

by

John Richard Liston

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
July 2004

Measuring User-Perceived Internet Performance in Multiple Locations

Approved by:

Professor Ellen Zegura
(College of Computing), Adviser

Professor Mostafa Ammar
(College of Computing)

Professor Jun Xu
(College of Computing)

Professor George Riley
(School of Electrical and Computer Engineering)

Dr. Michael Rabinovich
(AT&T Labs-Research)

Date Approved: July 29, 2004

To LuAnn,

For her love, support

and sense of humor.

ACKNOWLEDGEMENTS

This work could not have been completed without the help of many people. My advisor, Ellen Zegura, has been a constant source of motivation. Her ability to cut to the core of complex issues has never ceased to surprise and inspire me. Matt Sanders and Russ Clark have provided much appreciated friendship and stimulating discussion. Matt also participated in addressing the practical aspects of significant portions of the studies in this work. Thomas Hildebrandt and Bill Tice contributed to the design and implementation of very intricate, cutting edge pieces of software. I would also like to thank the faculty and students of Georgia Tech's Networking and Telecommunications Group — too many to name here individually — who have provided many thought-provoking discussions about many areas of networking research. Finally, I wish to thank my wife, LuAnn, for being patient, understanding and inspiring. While many relationships tend to be strained under the stress of Ph.D. studies, ours has only grown stronger.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Measurement studies of Internet performance are critical for validating or refuting widely held beliefs about Web behavior, and for shedding light on unknown behaviors. Results from these studies can guide Internet architects in making decisions that affect Internet Service Providers (ISPs), content providers and end-users. Examples of decisions that can benefit from measurement information include provisioning network capacity, placing Domain Name System (DNS) and Web servers, and tuning parameters of transport layer protocols.

Internet protocols and services may exhibit different performance characteristics when observed from different locations in the Internet topology; to date, however, there has been little work investigating the differences in these characteristics from multiple vantage points. Typically, performance studies present results of measurements taken in only one or two locations. Some of the reasons for the lack of work in this area are the following. First, performance measurement was not a high priority of Internet design and was not built into the network architecture. Second, it is difficult to obtain the necessary level of privilege at many different locations in the Internet topology to perform measurement studies. Finally, high expectations for real-time Internet performance is a relatively recent phenomenon.

In this thesis we develop several methods for gathering Internet performance data from multiple locations throughout the world, and to analyze data gathered. Our focus is on the protocols and services that support the World Wide Web.

In the first method we utilize a modified Web proxy. Our proxy captures and logs fine-grained performance information on a per-user basis. Our second method is to create and deploy a measurement package for examining DNS performance. We modified the BIND DNS server and packaged it with a script to drive the data collection. Our final method is to create and distribute an application to be run at user sites worldwide. One of the primary

tasks of the application is to provide performance data from each instance of the application executing at locations throughout Internet topology. We can use the information provided by this application to examine user-perceived Internet performance throughout the globe.

CHAPTER I

INTRODUCTION

Measurement studies of Internet performance are critical for validating or refuting widely held beliefs about Web behavior, and for shedding light on unknown behaviors. Results from these studies can guide Internet architects in making decisions that affect Internet Service Providers (ISPs), content providers and end-users. Examples of decisions that can benefit from measurement information include provisioning network capacity, placing Domain Name System (DNS) and Web servers, and tuning parameters of transport layer protocols.

Internet protocols and services may exhibit different performance characteristics when observed from different locations in the Internet topology; to date, however, there has been little work investigating the differences in these characteristics from multiple vantage points. Typically, performance studies present results of measurements taken in only one or two locations. There are several reasons for the lack of work in this area.

The first reason is that performance measurement was not a high priority of Internet design, so it was not built into the network architecture. The Internet is comprised of many interconnected machines that implement a common set of standardized protocols. There are many decisions that must be made when designing a network architecture. The fundamental goals of a network will determine how these choices are made. David Clark, one of the principal Internet architects, writes in a SIGCOMM 1988 paper that the top fundamental goal of the Internet was "to develop an effective technique for multiplexed utilization of existing interconnnected networks" [15]. In this paper network performance measurement is never explicitly mentioned. Also, the second level goals for which measurement must play some part — resource management, cost effectiveness and accountability — are low in the list of priorities. This is not to imply that network measurement was not a consideration.

In fact, the first Interface Message Processor, or IMP, was installed at the Network Measurement Center at UCLA [27]. It does mean, however, that if network measurement is to be performed then new techniques must be developed.

Secondly, it is difficult to obtain the necessary level of privilege at many different locations in the Internet topology to perform widespread measurement studies. There are many tools that allow network operators and managers to locally examine and manipulate network traffic. These tools — `traceroute`, `tcpdump`, `dig`, `ping` and others — as well as Web server logs are often used in measurement studies. However, they are rarely used in studies that claim to characterize user-perceived performance. Furthermore, studies that have been successful in collecting data from multiple locations do not generally claim to characterize user-perceived performance. This is because such data is usually obtained from endpoints in well-connected locations — research institutions rather than commercial ISPs.

A third reason is that high expectations for real-time Internet performance is a relatively recent phenomenon. The World Wide Web, or simply the Web, came into existence in 1993 with the appearance of the Mosaic Web browser. Prior to 1993, Internet applications were relatively forgiving of periods of congestion and poor performance. Examples of these applications are email, IRC (Internet Relay Chat) and gopher. Since about 1996 the Internet has become so ingrained in world culture that for the forseeable future any service intended to reach the widest possible audience must have an Internet presence. This has come to include services like real-time stock quotes, instant news, audio/video streaming and online merchandising. These services demand both reliable and responsive real-time performance — demands that were not original requirements of the Internet design. The success of these services is highly dependent on sufficient user-perceived network performance.

This thesis describes different methods to examine the performance of Internet protocols and services both in isolation and in tandem. We are primarily concerned with evaluating the effectiveness of different methods for gathering data from multiple locations in the Internet topology. We will also provide analyses of the data collected using the methods and tools we develop.

We study these interactions using the following techniques:

- **Using a single proxy at the user.**

  We modified a proxy to capture Web performance information at the user. When the proxy is installed on a user's machine it logs each object being requested, the type of the request (e.g., for an image, script or html page) and additional information that aids performance analysis. We describe our method of grouping objects into pages, some problems that arise during this process and our solutions to these problems. We conclude with an evaluation of our method. This method is discussed in Chapter 4.

- **Building and distributing a measurement package.**

  Previous work illustrates that resolution of domain names can be a significant component of downloading Web pages. The actual process that occurs while performing the resolution is, by design, "hidden" from the user. We therefore built a tool that exposes this process and logs the performance information. Since we had some evidence that DNS performance at our site was not representative of all user sites, we packaged this tool with a script to drive the data collection and distributed the tool to users throughout the world. We then collected the data and performed a detailed analysis. The results of this study are presented in Chapter 5.

- **Building and distributing an extensible application that performs measurement.**

  The major obstacle to collecting data from users is the lack of incentive to participate; in general, individuals are not inclined to participate in a network performance study. In our next method we overcome this obstacle by tightly coupling the gathering of performance data with an application that users would be willing to run. In Chapter 6 we describe an application we developed and have made available to users worldwide. The application, called "What the Internet Sounds Like", or WISL, measures different characteristics of the network from the point at which it is running. During the process, the application plays sounds that represent the current state of the network. The sounds that are played can range from a literal notification of network state to an aesthetically pleasing aural representation of network conditions. During collection the data is collected and analyzed, providing a window into Internet performance that was previously not available. WISL is highly extensible; different network measurement modules can be incorporated into WISL, making it a suitable platform for network researchers to examine a variety of network characteristics from points throughout the network topology. WISL is described in detail in Chapter 6.

We discuss previous related work in network measurement in Chapter 2. Then in Chapter 3 we provide an overview of the process of downloading a Web page and of the operation of DNS. To provide a backdrop for some of our design choices we also discuss the pros and cons of measurement options with respect to Web performance. In the concluding chapter we summarize the contributions of this thesis.

CHAPTER II

RELATED WORK

Since performance measurement is not built into Internet design, the Internet poses interesting challenges for measurement. In this section we describe ongoing work in Internet measurement, with emphasis on measuring protocols and services that support the Web. We also discuss this work in the context of characterizing end-user measurement and worldwide measurement.

2.1 Web Performance

There have been many studies that examine the Web performance with the goal of explaining Web behavior and understanding how to best address the performance issues.

The technique used by Cunha, Bestavros and Crovella [20] is closely related to our proxy work. They instrumented the Mosaic Web browser to record the Uniform Resource Locator (URL), time of access and retrieval time for each object requested by the browser. This data collection work formed the basis for important studies in network performance [19]. Our instrumented Web proxy, described in Chapter 4, essentially extends the functionality of the browser to log similar information. The advantage of our approach is that it works with virtually any Web browser. Additionally, our proxy logs the time required to resolve the domain name of the Web server to which the request is sent, which can be a critical, but often overlooked, component of the download time. While they were successful in collecting data from many sessions and users, they were limited to a single site, whereas our data was collected in multiple sites.

Also similar in method to our approach, the Medusa proxy developed by Koletsou and Voelker [30] captures user traces. Their goal, however, was to replay the user sessions under various system configurations to compare the performance implications of different configurations.

Some studies infer user-perceived Web performance by examining traffic at the server. One early Web performance study by Balakrishnan, Seshan, Stemm and Katz [11] collected and analyzed packet-level traces from the primary Web site for the 1996 Summer Olympic Games. Their findings confirm the heterogeneity of Web performance at that time, and they demonstrate that hosts that are close to each other experience essentially the same throughput. We point out that all their findings must be considered with respect to their point of collection. Data collected in another location may report similar findings, but actual reported throughput and latency may differ significantly.

We mention also that Keynote [4] is a commercial system designed to provide their clients with strong comparative data about the performance of downloading their Web pages. They have installed an infrastructure of 1,500 measurement computers in 50 cities worldwide. While some of our goals are similar, our aim is to be able to publish our data for independent analysis and validation of our results. Also, in contrast, we work towards the goal of studying real user experience, whether or not the users visit any particular set of Web pages.

2.2 User-Perceived Internet Performance

There is some related work that, similar to ours, performs measurement on end-systems. Such pieces of work include NETI@home [39], ANEMOS [21] and ICPLD [2].

NETI@home is very similar in spirit to WISL. Like WISL, the goal of NETI@home is to collect network performance statistics from end-systems. Also like WISL, it accomplishes this by providing a software package that the user downloads and runs on the user's machine. The primary difference between NETI@home and WISL is in the incentives for users to participate in the data collection. The primary incentive for NETI@home users is altruism. The NETI@home home page asks users to download the package and run it where it will collect data and send it for subsequent analysis to a central server at Georgia Tech. By participating in the data collection process users will provide data to the researchers. NETI@home has had excellent response from the user community. In contrast to NETI@home WISL is intended to appeal more to the user's interest in the applications itself, rather than to participate in a performance study. The primary incentive for WISL users is that the package will generate music that users would like to hear, making it an interesting application in its own right. Another distinguishing characteristic of WISL is that WISL is highly extensible and useful to the networking community at large. At the network level researchers can contribute new measurement modules. In this way WISL provides researchers with the ability to perform measurements in parts of the network that were previously inaccessible. And at the highest level WISL allows for the incorporation of new SoundPalettes which map network-level events to sounds. This allows for incorporation of multiple SoundPalettes in the hopes that users will find one or more that suit their individual taste.

ANEMOS (Autonomous NEtwork MOnitoring System) is a system written in Java (for portability) that supports real-time processing, analysis and visualization of active measurements. WISL is closely related to ANEMOS in the sense that both systems use active measurements to detect network conditions and generate events — "rule-based alarms" in ANEMOS. Both systems are also extensible at the measurement layer. There are two primary differences in the two systems. The first is they have different intended audiences. ANEMOS is intended for execution by network researchers and network managers, whereas WISL is intended for execution by "average Internet users". The second difference lies in the method of data collection. ANEMOS has a centralized *Coordinator* module, which schedules the measurements and collects and analyzes the data. In contrast, data collection in WISL occurs whenever end-users execute the application; data is actively pushed to a central server for subsequent analysis.

ICPLD (Internet Connection Performance Logging Daemon) is a daemon that runs on an end-system. It sends ICMP requests to a user-specified IP address. The goal is to provide real-time monitoring of the existence of a network connection between the two machines. It timestamps and logs both failed attempts and the first successful reply after failure. There is, however, no attempt to collect and compare data from multiple locations.

2.3 DNS

Relative to other measurement efforts there have been surprisingly few DNS performance studies. Some existing studies have focused on performance from the point of view of root servers. Danzig et al. [22] analyzed logs from one root nameserver and three nameservers that replicated various domains. The focus of their work is on the consistency and correctness of portions of the DNS database. Similarly, Brownlee et al. [14] analyze logs from a single root server, reporting on the apparent problems in local name servers that send queries to the root servers. In contrast, our work examines the process of resolving domain names from the point of view of the user, focusing on overall response time and its impact on user-perceived performance. We show that root server performance in fact has very little impact on user-perceived DNS performance.

Work done by Jung, Sit, Balakrishnan and Morris [28] is closely related to ours in that it focuses on user-perceived DNS performance. They captured traces of real user traffic from two sites — MIT and Kaist — and analyzed the impact of caching on user performance. In our work we illustrate that for certain measures the variance in performance can be so high that, depending on the particular performance characteristic being observed, it may be necessary to take measurements in many locations to be able to generalize the results.

The goal of work by Shaikh, Tewari and Agrawal [38] is to explore the impact of low TTLs on user-perceived Web performance. They measured DNS lookup times using nameservers in four locations: Masssachusetts, Michigan, California and New York. They found that caching reduces median lookup times by more than two orders of magnitude.

Cohen and Kaplan [16] propose the prefetching of DNS resolutions as one of three methods to reduce user-perceived latency. Their data are based on measurements of lookup times from three locations: AT&T, Stanford and Tel-Aviv University. In another paper [17] they propose several methods to reduce DNS latency by refreshing expired entries in the DNS cache.

Wills and Shang [42] investigate the impact of DNS on user-perceived Web performance. They find that only 20% of lookups are not cached at the LDNS and that only 20-30% of those non-cached names take more than one second. Their results are based on sampling the response time at their site for 100 popular domain names and 100 domain names drawn randomly from NLANR Web cache logs.

Although both the King system [26] and work by Wills, Mikhailov and Shang [43] leverage the DNS system for measurement purposes, these studies does not characterize DNS performance. We will therefore treat their work as measurement infrastructures and consider them in Section 2.4.

2.4 Wide-Area Measurement Infrastructures

In recent years there has been significant movement towards measurement infrastructures that allow researchers to gather high-quality Internet performance data. Mahdavi, Paxson, Adams and Mathis created the National Internet Measurement Infrastructure, or NIMI [36], which was a first step towards this goal. In this work they deployed a daemon on machines located in multiple sites — 35 sites were reported in [35] — and coordinated measurements among them. Our work differs from theirs in that we seek to deploy measurement tools on end-user machines and make no attempt to coordinate measurements, although our architecture allows limited coordination of measurements.

While it is not a measurement infrastructure per se, Allman, Blanton and Eddy [10] describe a system for sharing network measurements. This has the potential to make it possible for researchers to use data collected by others in multiple locations to perform a wide-area performance study.

PlanetLab is a collection of machines distributed throughout the globe. It allows an application to be run across some or all of the machines. Many types of application are possible, including those that perform network measurement. Scriptroute [40] is a scripting system for performing and coordinating measurements across multiple locations. Scriptroute is currently deployed on some set of PlanetLab sites. It is quite possible for PlanetLab nodes to be installed in a very large number of locations in countries throughout the world. One potential drawback that we are attempting to overcome in our work is that the nodes must be continuously available, and as they tend to be geared towards the networking research community, they are likely to be installed at primarily research facilities. For certain types of measurement this may be quite sufficient. The study by Akella, Seshan and Shaikh [9] used twenty-six PlanetLab nodes to probe for wide-area Internet bottlenecks. The high speed access provided by PlanetLab was an important characteristic of their study. For other measurement studies, however, the PlanetLab infrastructure may not be generally representative of user-perceived performance.

King [26] is both a tool and a service. The tool attempts to identify the two DNS servers that are closest to two given endpoints and causes a query to be sent from one to the other. The service uses this tool to estimate the round trip time between the two endpoints. King could be used as the basis for a very wide-scale study about Internet topology.

Wills, Mikhailov and Shang [43] also leverage DNS to perform measurements. They send requests to twenty different local DNS servers (LDNSs) to infer the popularity of data servers. This is an intriguing technique that could be exploited for very wide-scale measurement studies.

WebPerf [7] is a tool that is intended to be installed at many sites worldwide and run continuously. There do not appear to be any measurement studies published that are based on Webperf so there is little public information about how the tool actually works. It is written in perl and appears to send a URL from a central location to all locations where the tool is currently running. It therefore appears as though it has potential to form the basis for an interesting global Web performance study.

E2E piPES [1] is an Internet2-specific measurement system that aims to allow arbitrary users to locate and report network problems. Examples of information that can be gathered include loss, jitter, one-way latency, flow data and "SNMP-queryable" router data. This information can be determined on the granularity of a single link in a path.

Surveyor [29], a project of Advanced Network & Services and the Wisconsin Advanced Internet Laboratory, currently consists of about 70 nodes performing one-way delay and loss monitoring. Each node is a dedicated machine that provides a higher degree of security and control over the system load. It utilizes specialized GPS-enabled hardware to synchronize system clocks. The system is in the process of being enhanced to provide more generality.

Enable [41] is a system for adaptive monitoring, publishing and analysis of throughput and delay values. The primary goals for the system are to provide manageability, reliability, and adaptability for high performance applications running over wide-area networks. The main component is a network advice server that may reside on the host of any data source. The server can be queried by clients to determine the optimal TCP buffer size for a given path.

CHAPTER III

BACKGROUND

In this thesis we focus on protocols and services that support the World Wide Web. Primarily we are interested in HTTP and DNS, both of which are required for Web browsing sessions. This chapter explains the operation of these protocols and services relevant to our work.

3.1 Web Session Overview

We begin with an overview of the sequence of events that occurs when a user downloads a Web page. The process begins at the user/browser interface. The user indicates to the browser the Uniform Resource Locator, or URL, of the page he or she wishes to load through one of the following means[1]:

- The user may enter either a complete or partial URL into a text field provided by the browser. In the case of a partial URL, the browser follows a fixed set of heuristics to construct a complete URL. The heuristics may vary among browsers.
- The user chooses a URL from a set of "bookmarks" or "favorites" — URLs previously stored by the user for future use.
- The user clicks on a hypertext link or an "action button" in a currently displayed Web page.

The normal actions taken next are illustrated at a high level in Figure 1. Once the browser has obtained the desired URL it parses the URL to determine the domain name or IP address of the server that is the target of the request. Although the server's IP address may be indicated directly in the URL, most commonly the URL contains the server's domain

[1]Websites can cause automatic reloading of a Web page via the META tag's http-equiv="refresh" attribute. In this case, no user action is required.

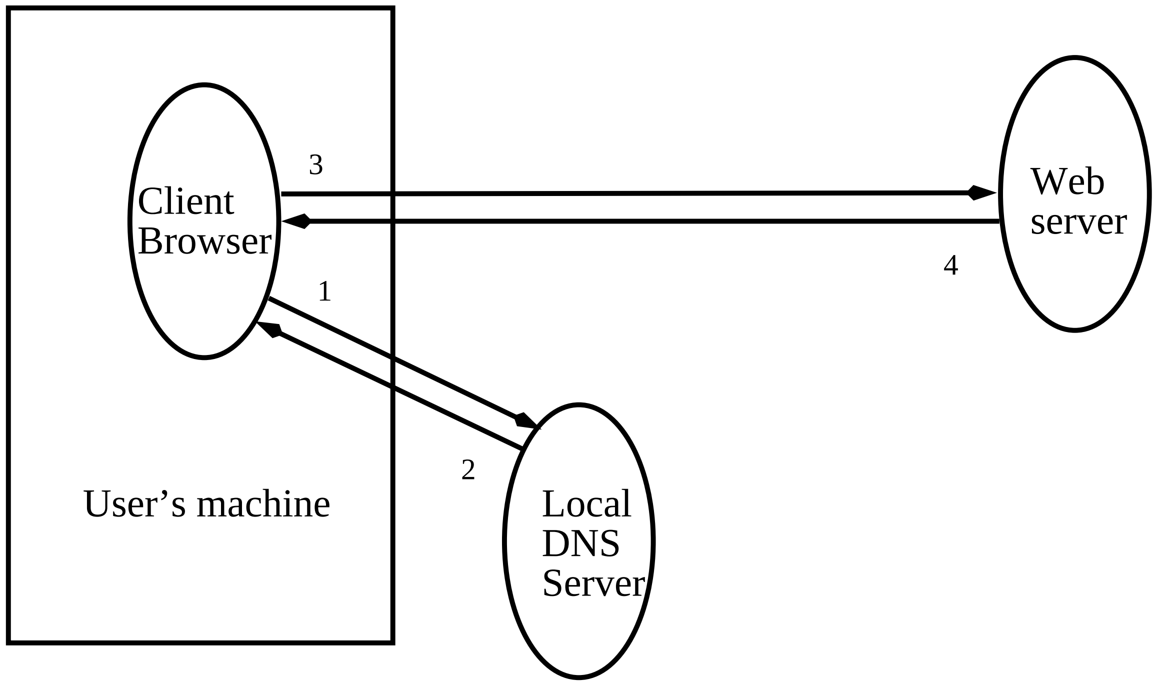


Figure 1: Normal loading of a web object.

name. The browser (1) sends a request via UDP to the local DNS server, which resolves the domain name on behalf of the client. This process is discussed in detail in Section 3.2. The DNS server (2) sends the response to the browser.

The browser then (3) sets up a TCP connection and sends an HTTP request to the Web server to (4) retrieve the initial object, or *base page*, indicated by the URL.

The base page may contain tags indicating additional objects that must be loaded to complete rendering the page. Examples of these objects are images, frames and scripts. These objects may reside on servers other than the one where the base page was retrieved. The above process is repeated for each object until there are no more objects to load for the page or until the user cancels retrieval of the page.

3.2 Domain Name System

The Domain Name System (DNS) is a distributed, dynamic, hierarchical database primarily used to resolve the human-readable domain names of remote machines to IP addresses. This resolution, called a domain name *lookup*, is typically the initial step in communication between two IP endpoints when the remote IP address is not known. Thus, DNS is a critical component of the operation of many Internet applications, including Web browsers.

Excellent descriptions of the operation of DNS are provided in several papers [22, 28, 25, 38, 18], and the details of this process are specified in the associated RFCs [33, 34, 23]. We will illustrate here in detail how the DNS structure and operation affect user-perceived performance. We will also highlight aspects of the DNS architecture that may cause clients in different locations to experience very different performance even though they are resolving the same name.

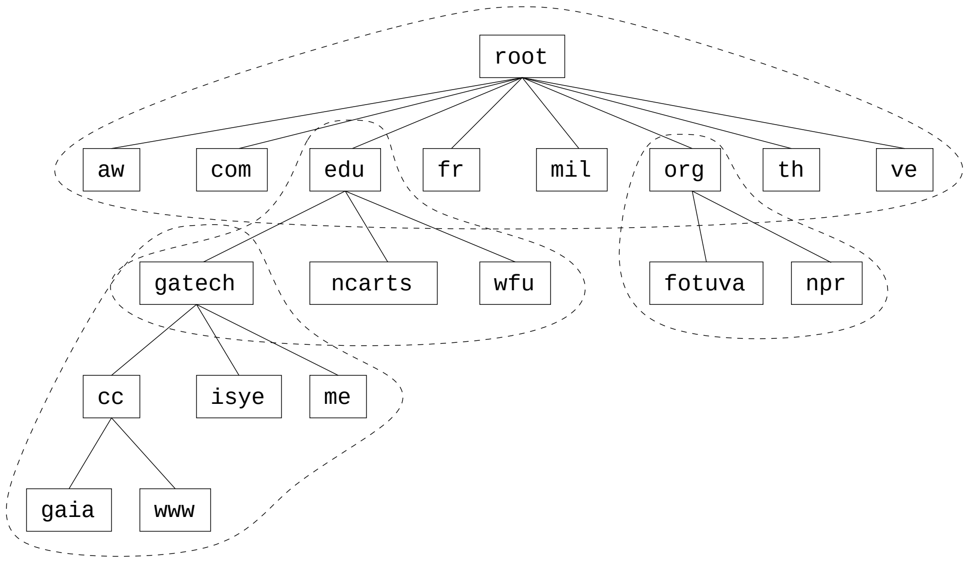


Figure 2: Structure of the DNS namespace.

The DNS system is structured to simplify administration of different portions of the namespace, or domains. In this way IP addresses can be easily mapped and re-mapped by local administrators without having to notify a central authority. The organization of the namespace for domain names has a hierarchical structure, where each node in the tree has a label. As an example, a small portion of the namespace tree is depicted in Figure 2. The root of the namespace is indicated simply by a dot (.). Domain names are built up by walking down the namespace tree and at each node in the tree, prepending the label contained in the node with each step. A dot is also used to separate each label. The names at the leaves of the namespace tree are typically the domain names of actual machines. Names in the tree that are not leaf nodes are the names of domains. In some contexts it is not necessary to specify all of the labels in a domain name. A **fully qualified domain**

name, or FQDN, is a domain name in which all labels are specified and that ends with a dot indicating the root domain. For example, the name `gaia.cc.gatech.edu.` is the FQDN for a machine that resides in the domain `cc.gatech.edu`.

DNS allows administrators to map multiple names to the same IP address. This is accomplished with CNAME records. A CNAME record indicates that the current domain name is actually an alias, and maps it to a different domain name which is is the canonical name. Administrators can also map a single domain name to multiple IP addresses. When this happens the client software chooses one of the IP addresses — typically the first one in the set.

The DNS database is partitioned into zones. A **zone** is a contiguous section of the namespace tree that is managed by a single organization. The zone structure allows for local administration of the database. Subtrees of the namespace may be delegated to other organizations by creating a new zone. Figure 2 also shows four administrative zones, indicated by dashed lines. The zone at the top of the namespace tree is the root zone. The root zone delegates all other zones by simply pointing to the nodes at the top of its subtrees. In the figure we expand slightly two of the many zones at this level: `edu` and `org`.

The process of resolving a domain name works as follows. So that each application does not have to implement its own DNS resolution functionality, applications incorporate DNS resolver libraries. When an application needs a domain name resolved, it invokes the resolver function. The resolver is configured with the IP address of its local DNS server (LDNS), so it sends a sends a DNS query over UDP to the LDNS. This is typically a recursive query: a request for the LDNS to perform the resolution on behalf of the client. The LDNS requires a starting point for performing resolutions in the DNS namespace, so it is configured with the IP addresses of the thirteen DNS root servers. The LDNS sends a query to one of the root servers. Assuming the query is valid, the root server will respond with an NS record. NS records are used to inform the client — in this example the client of the root server is the LDNS server — that it must query another nameserver that will be closer to the target of the query. Responses that contain NS records are known as "referrals". An LDNS may receive multiple referrals before finding an authoritative nameserver that can answer the query. The LDNS may encounter the following responses from the authoritative nameserver:

- an A record that contains the IP address of the domain name
- an indication that the name is invalid
- no response because the nameserver is not available
- no response because either the UDP query or response packet was dropped

In the first two cases the query is satisfied, and the LDNS sends its response to its client. In the last two cases the LDNS sets a timeout. Upon expiration of the timeout the LDNS resends the query to the nameserver. After a configurable number of timeouts the LDNS gives up and sends a failure response packet to its client.

One characteristic of the organization of the DNS database is that the partitioning of the zones does not necessarily dictate how and where the database is actually stored — only how the data is located. It is the responsibility of the zone administrators to decide where to store the zone data. The RFCs provide some guidelines for how to organize and serve the data so that the system is uniformly robust and reliable. As an example, one RFC states that a "given zone will be available from several name servers to insure its availability in spite of host or communication link failure. By administrative fiat, we require every zone to be available on at least two servers, and many zones have more redundancy than that [33]."

3.2.1 Differences in DNS Performance

This section illustrates how two clients can experience very different performance during normal DNS operation while resolving the same name. Figure 4 shows an example of three different clients resolving the same name: `bigtooth.cc.gatech.edu`. The two clients on the right side of the figure share the same LDNS server. This figure emphasizes the fact that two resolutions of the same name can result in the queries being sent to a completely different set of servers.

We begin with Client 1, which sends a recursive query to its LDNS. The LDNS has no information cached about `bigtooth.cc.gatech.edu`, `cc.gatech.edu`, `gatech.edu` or `edu`.

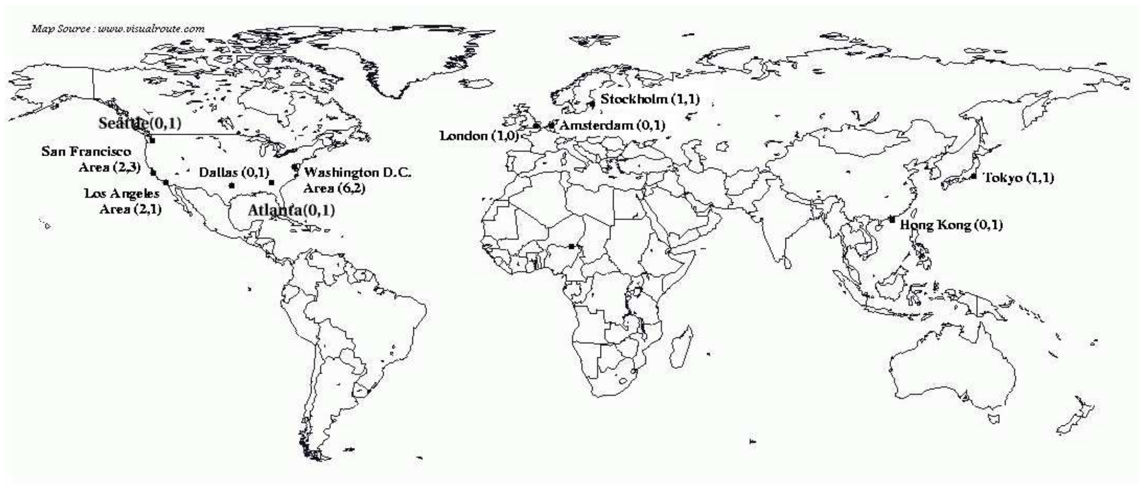


Figure 3: Locations of root and gTLD servers.

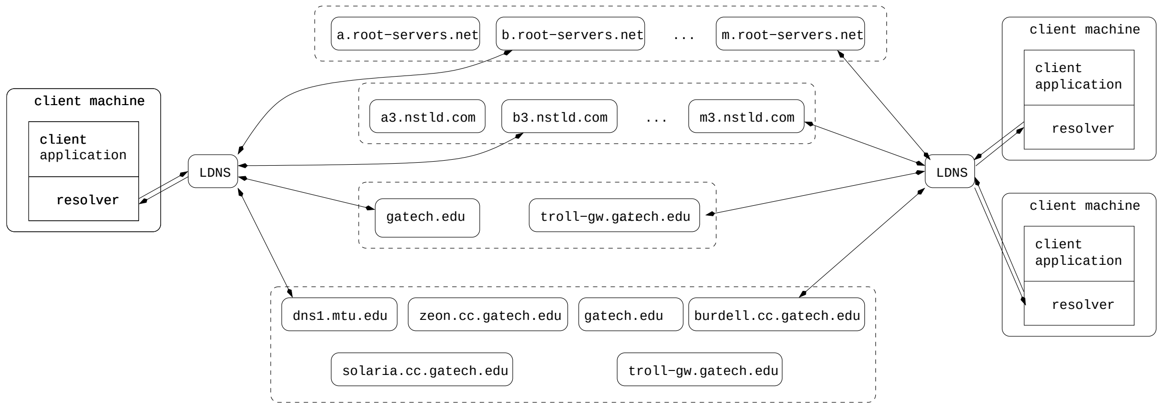


Figure 4: Example of resolving a domain name.

Its first request is therefore sent to one of the root servers. It arbitrarily chooses B.ROOT-SERVERS.NET. The response to this query is a referral indicating that the client must now query one of the thirteen nsTLD servers. Client 1's LDNS chooses B2.NSTLD.COM for this query[2]. To increase reliability, it is recommended that domain information be replicated at different servers, and that they "be placed at both topologically and geographically dispersed locations on the Internet" [24]. So in our example, the response is a referral to

[2]From the web page describing djbdns [13]: "*dnscache* simply contacts a random server, to balance the load as effectively as possible. BIND keeps track of the round-trip times for its queries to each server, with various bonuses and penalties, and then sends all its queries to the 'best' server".

one of two servers that should have the answer: `gatech.edu` or `troll-gw.gatech.edu`. Client 1's LDNS chooses `gatech.edu`, sends the query and receives another referral to the six nameservers at the bottom of the figure. It chooses `dns1.mtu.edu`, sends the query and receives the answer. It then sends the response to Client 1. Note that all information it receives about the namesspace tree has been cached. Each response contains a time-to-live (TTL) indicating how long the client may cache the answer.

Client 2 sends a query to its LDNS for the same name. Many of the same steps are taken as for Client 1, but we see that a completely different set of servers is queried: `m.root-servers.net`, `m3.nstld.com`, `troll-gw.gatech.edu` and `burdell.cc.gatech.edu`.

Client 3 then sends a query to its LDNS, shared with Client 2. Assuming the query is sent within the TTL of Client 2's previous response the LDNS still has the answer cached and immediately sends the response to Client 3.

Figure 3, obtained from CAIDA [14] and updated to reflect two additional gTLD servers in Atlanta and Seattle, further illustrates the potential for clients to experience different DNS performance depending on location. The figure shows the locations "of the root nameservers and gTLD servers. The (x,y) notation near the city names indicates the number of root servers (x) followed by the number of gTLD servers (y) in that area. Notice the large number of both types of servers around Washington D.C. and in California." [14]. The map highlights the fact that the root and gTLD servers that are central to the operation of DNS are geographically concentrated in the U.S., with many geographic regions entirely unrepresented.

We must emphasize that the DNS is not a static entity. There is much current activity in extending DNS to provide security and add additional features. Any changes will have effects on user-perceived performance that will warrant further measurement study. In fact, since the time of our study some of the zones high in the DNS hierarchy (`com`, `org`, `edu`, etc.) have been restructured.

3.3 Measurement Options

We conclude this section of background with a discussion of options for performing user-perceived measurement. We discuss the advantages and disadvantages of each option and illustrate where our work lies.

It is possible to analyze Web performance using packet traces or access logs provided by an ISP. However, getting ISPs to cooperate with measurements may be challenging. There are several reasons for the difficulty. First, releasing packet traces may expose sensitive information of clients, like passwords, credit card numbers and other personal information. ISPs are understandably reluctant to provide external access to such traces. Second, setting up measurement infrastructures may require a significant time commitment from ISPs. System and network administrators' time is often scarce even for normal day-to-day activities. Getting participation on projects in which there are no apparent immediate benefits for their clients is unlikely.

Another option is to collect data inside the network. This option takes several forms. Examples of data collection in this class are: packet traces of data traversing a single AS; probing one or more ASes; or performing active measurement using existing measurement infrastructures. Certain studies will benefit from this approach. Examples are those that explicitly examine characteristics of the network that affect user traffic in aggregate like congestion patterns, network structure, routing and traffic patterns. Studies that use this option do not report on the actual user experience.

Another option may be to bypass the ISP infrastructure by obtaining user accounts in multiple domains. However, taking measurements directly in this fashion may be prohibitively time-consuming and expensive. Even if we were able to obtain a large number of user accounts throughout the world, results that are taken in a central location claiming to represent a user's experience in a remote part of the world would be questionable.

The final approach, and the one used in this thesis, is to take measurements on multiple users' host machines. This approach will provide data that truly reflects characteristics of the network as experienced by the user. This data can either validate or refute claims made in other studies. Either outcome will be a significant step in network measurement.

CHAPTER IV

USING A PROXY TO MEASURE USER-PERCEIVED WEB PERFORMANCE

In this chapter we present our first method of collecting user-perceived web performance data: using a specially instrumented proxy. We then discuss challenges in capturing the performance of downloading a Web page at the client, and describe our solution.

4.1 Requirements

Our goal is to create a measurement tool that logs information about the performance experienced by the user. Our criteria for the measurements are as follows:

- Taken from real user behavior. We want to collect measurements during normal user operation.
- Captures individual aspects of loading each object such as DNS resolution time, timing of each request coming from the browser, and time for complete objects to be downloaded.
- Transparent to users. Since we will make measurements during normal operation, we want to minimize any performance degradation or other artifacts from measurement that might alter user behavior.
- Collected in a range of environments, broadly defined to include a variety of operating systems, browsers, access technologies, etc. This is to ensure that any local effects such as an atypically fast connection do not skew our conclusions. As a corollary, we require that the measurement technique require no special privileges (e.g., root access) on the part of the users participating in the study.

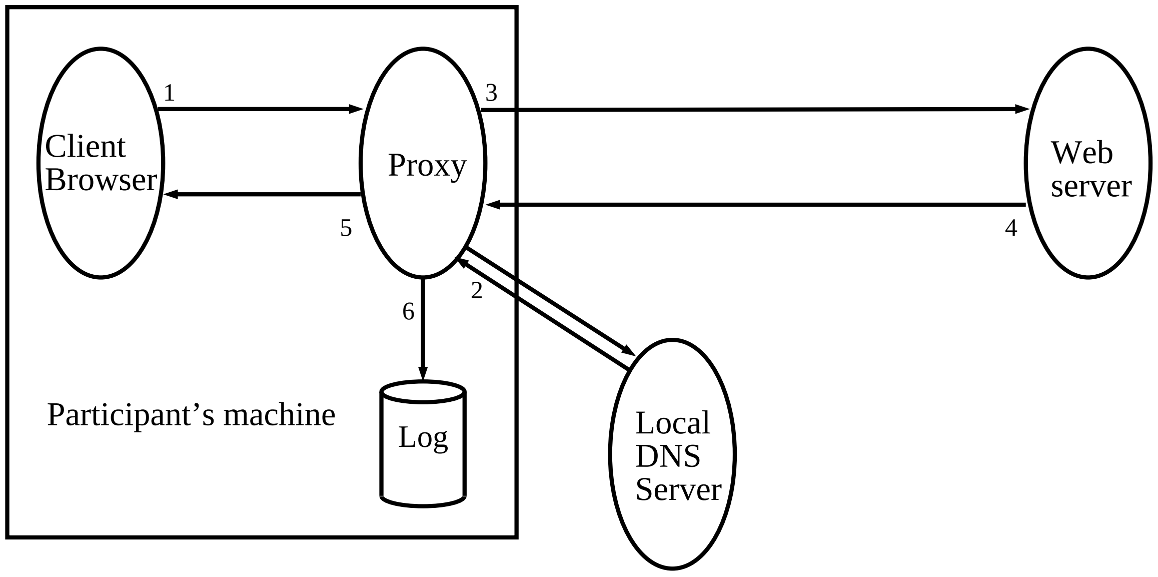


Figure 5: Sequence of events in our measurement architecture.

4.2 Method Overview

To satisfy our objectives and overcome the obstacles, we implemented data collection in a web proxy with one proxy instance per browser instance, as illustrated in Figure 5. It is not strictly required for the proxy to execute on the same machine as the browser, but doing so minimizes the effects of competing network traffic on the measurements. As illustrated, the proxy is situated between the browser and any web servers accesses; it also handles all calls for DNS resolution.

The proxy operates as follows: (1) intercepts client requests, (2) performs the DNS resolution, (3) connects to the target server and sends the request to the server. The server then (4) responds to the proxy with a header and (typically) data for the requested object. As the proxy receives the responses, it (5) passes them to the client. While the proxy passes headers and data in both directions, information of interest is parsed from both the headers and the data and (6) gets logged to a log file. We post-process the log file to analyze the data.

Each line of the log file contains the following information:

- **ID:** An identifier which is unique to each object being requested by the proxy.

- **TYPE:** A string indicating the type of the line. The complete list of TYPES is given in Appendix A. The use of some of these TYPES is highlighted in Section 4.3 where we describe details of the proxy design.

- **ADDITIONAL INFO:** Each line type may cause additional information to be logged. For example, the actual IMG tag is logged on an IMG line type. Some line types, such as an **EOH** line, do not have additional info.

Although the browser must maintain explicit information specifying how objects are grouped together into pages, it does not in general indicate this in its requests. Therefore, from the point of view of the proxy, each request is independent; the proxy cannot maintain or log this information. Often, however, the variables we wish information about depend on how the objects are grouped together into pages. Therefore, one of the most important functions to be performed during post-processing is to reconstruct how the objects are grouped to form a complete web page.

In Figure 6 we illustrate a simple example of how we perform this grouping. Figure 6-A depicts the proxy's view of objects being transmitted over time. The horizontal axis is the time line, and the thick horizontal lines depict the time from the initial receipt of the request for the object at the proxy to the time the object has been completely transmitted to the browser. On the vertical axis the objects are numbered in the order in which they are requested. Not pictured is the fact that along with information about the timing of the objects, the logs also contain the URLs of the objects and any information about objects which may be requested as a result of receiving this object.

Figure 6-B depicts the post-processing, which occurs in two steps. First, for each object, we search the set of objects retrieved to date for a tag which caused this object to be retrieved. If one is found, we add an arrow extending from the retrieved object to the one that contains the tag.

When the complete log file has been processed, we have all the information necessary for grouping objects into pages. Figure 6-B also depicts this final stage in post-processing. Any object which does not have an arrow emanating from it is assumed to have been requested

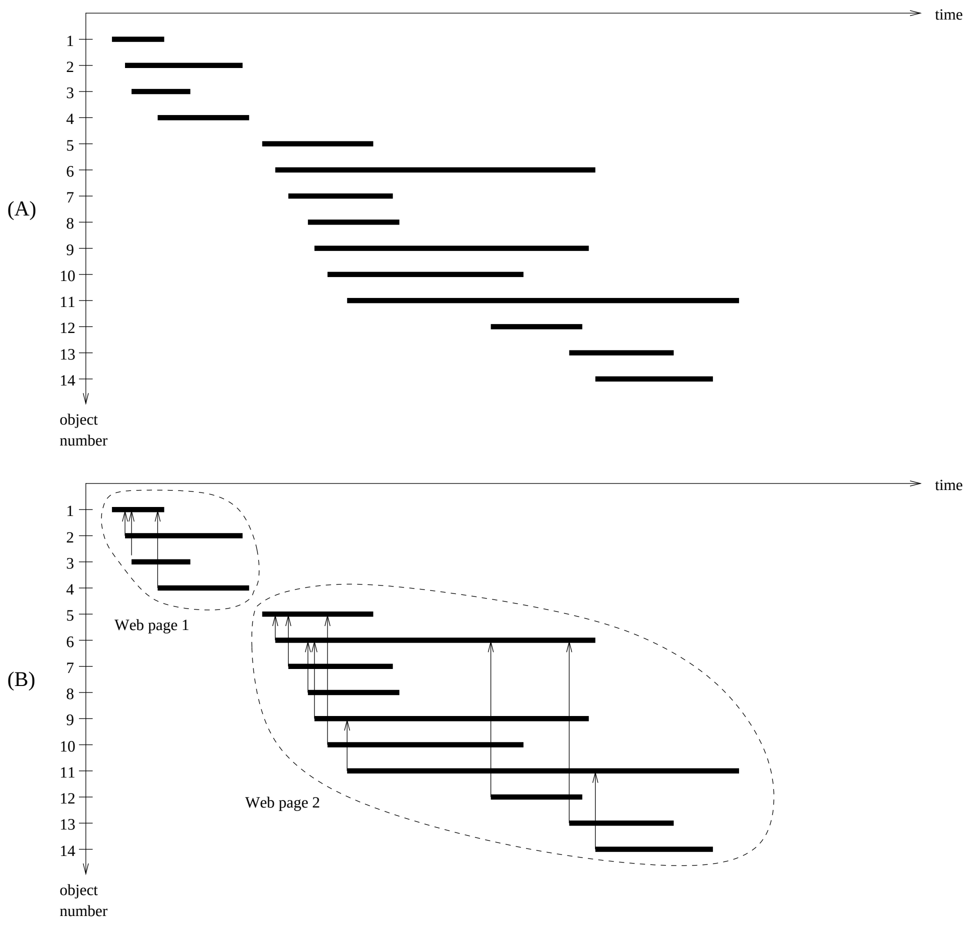


Figure 6: Grouping objects into web pages during post-processing

by the user. These are the base pages. We then group objects into a web page by associating with each base page all objects which can reach the base page by following a path along the arrows and objects towards the base page. Two such groupings are depicted in the figure.

This discussion describes the simplest case of post-processing. Issues which cause difficulties and the effects of these issues on our design are discussed in the next section.

4.3 Proxy Design Issues

In this section we discuss issues affecting the design and implementation of the proxy. These issues determine what information was actually chosen to be logged in the log files.

We group the issues affecting our design into three categories. First, we discuss the issues related directly to the HTTP protocol itself. These issues are involved with information in the HTTP headers of the request and the response. Next, we discuss the issues that relate to the actual content of the objects. Finally, we discuss issues that arise from interactions between the proxy and browser.

4.3.1 HTTP Issues

Content-Encoding The HTTP protocol allows the data in a response to be encoded using different encoding schemes. Requests may contain an "Accept-Encoding" field in the request header that indicates to the server what encoding types the client is willing to handle. The server informs the browser via the "Content-Encoding" header field how the data must be decoded. For example, the server can deliver a gzipped HTML page accompanied by the header field "Content-Encoding: gzip". This, however, causes all HTML in the transferred object to be unreadable by the proxy since the data will not be decompressed until it is received by the browser.

To alleviate this problem, the proxy intercepts the "Accept-Encoding" field sent by the browser and changes it to "Accept-Encoding: identity; q=1.0, ∗;q=0". This causes the server to decode any encoded content, if possible, before sending. The proxy can then examine and parse the plain HTML. Doing so will cause a change in the timing of the rendering of the page. However, during development of the proxy we noted few situations in which servers provided compressed HTML objects.

Location The response header for an object may include a "Location" field. This field indicates to the browser that the object being requested is found at an alternate URL, and provides the URL for the browser to automatically query. There may be more than one location indirection that must be followed before the request for the object is satisfied. The new URL may contain the same domain name as the initial query, a completely different domain name, or it may point to a different name within the same zone as the original query.

The proxy logs the "Location" header field, and in post-processing, objects are linked to previous requests in which the response header had a matching "Location" field.

Proxy-Connection Our proxy implements HTTP/1.0. It therefore does not handle Keep-Alive connections. Specifically, it does not handle more than one set of headers sent from the client to the server on a single connection. The proxy handles this by removing the "Proxy-Connection: Keep-Alive" from headers sent to the server. This forces the server to close the connection after the requested object has been sent. It has the effect of requiring a separate process to be created for each requested object, each with a separate tcp handshake.

Content-Type Objects with certain content-types — for example, images — will never generate subsequent requests from the browser. As a performance enhancement the proxy examines the content-type of the object as it is received from the server. If it will not cause a subsequent request to be sent, the proxy passes the data directly to the browser without parsing it.

Also, if the content-type is application (e.g., application/x-javascript or application/x-shockwave-flash), then we assume that another page caused it to be loaded. In this case, if no other heuristics have determined which base object this object should be linked to, we link it to the most recently identified base object.

4.3.2 Content-Driven Issues

JavaScript JavaScript is a scripting language that allows for development of web applications. Client-side JavaScript is dynamically interpreted by the browser and can cause the browser to retrieve remote objects. It is impossible for a proxy to predict with 100% accuracy what a request from a browser will look like when the page includes JavaScript and the browser has JavaScript turned on. A simple example of the problem is illustrated by the following JavaScript code:

```
⟨script lang="javascript"⟩
NumR=Math.floor(Math.random()*10000000);
document.write('⟨IMG SRC="http://foo.bar/' + NumR + '"⟩');
⟨/script⟩
```

In the above case, the proxy and the browser will choose different random numbers and the resulting IMG tags will not match. The post-processor will be unable to match the incoming request with the page causing the request, so it will incorrectly interpret the new request as a base page. We handle this ambiguity by causing delimiters to be placed

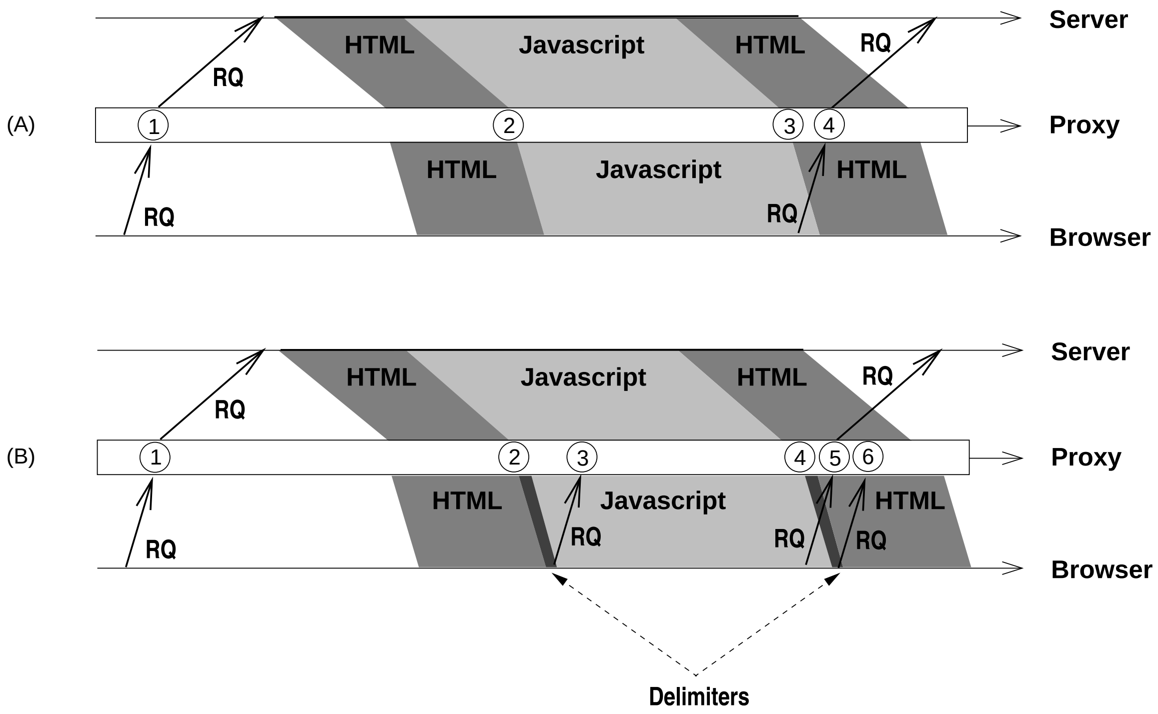


Figure 7: Identifying requests generated by JavaScript

in the proxy's log when a section of JavaScript has been encountered (see START_JS and END_JS in Table 1). During post-processing, any object that occurs between the delimiters is assumed to have been generated by the JavaScript section of the previous object.

As illustrated in Figure 7 the delimiters cannot be logged while the stream is being received from the server since the timing of the proxy's parsing of the JavaScript tag and the subsequent browser requests are not synchronized. The figure shows two different timing diagrams of events seen by three entities: the browser, proxy and remote server. In Figure 7-A, the browser generates a request to the proxy which, at (1) is logged and forwarded to the server. The server responds with an object which contains HTML and JavaScript portions. At (2) the proxy logs START_JS to indicate the start of JavaScript code and at (3) it logs END_JS to indicate that the end of JavaScript code has been reached. However, the request generated by the JavaScript code is not seen by the proxy until (4), which occurs outside the delimiters.

The delimiters can be properly placed as shown in Figure 7-B. The idea is to cause the browser to send well-known requests to the proxy indicating when the JavaScript has begun

Table 1: Log file line types.

Line Type	Description
ENTER	Provides a timestamp when the proxy first receives a request from the browser.
REQ	Logs the actual request line sent from the browser.
HOSTNAME	Name of the machine in the URL of the requested object, and the time it took for the name to be resolved.
EOH	Indicates that the proxy has received and parsed the complete header.
META	Indicates that the received object contained a META tag. META tags may contain a "url" attribute causing another object to be requested.
BODY	Indicates that the received object contained a BODY tag. BODY tags may contain a "background" attribute causing another object to be requested.
IMG	Indicates that the received object contained an IMG tag. IMG tags are required to contain a "src" attribute which may cause another object to be requested.
INPUT	Indicates that the received object contained an INPUT tag. INPUT tags may contain a "src" attribute causing another object to be requested.
TD	Indicates that the received object contained a TD tag. TD tags may contain a "background" attribute causing another object to be requested.
TABLE	Indicates that the received object contained a TABLE tag. TABLE tags may contain a "background" attribute causing another object to be requested.
EXIT	Provides a timestamp when the proxy has completed handling the request.
ELAPSED	Provides a calculation of how long the proxy took to load this object.
APP	Indicates that the received object is an application as specified by the **Content-Type** header field, and will not cause another object to be requested.
APPLET	Indicates that the received object contained an APPLET tag. APPLET tags may contain a "src" attribute causing another object to be requested.
FRAME	Indicates that the received object contained a FRAME tag. FRAME tags may contain a "src" attribute causing another object to be requested.
LOCATION	Indicates that the header for the requested object contained a **Location** header field. Upon receiving this header field, browsers immediately make a request to for the indicated object.
REFERER	Indicates that the header of the request contained a **Referer** header field. This field can indicate which object caused the immediate request for the object, or it can indicate the page that contained the link the user selected in order to retrieve the object.
START_JS	Provides an indication that the browser has begun interpreting a section of javascript. See section 4.3.2.
END_JS	Provides an indication that the browser has completed interpreting of a section of javascript. See section 4.3.2.

and ended interpretation. The delimiters are logged when these requests are received by the proxy. Object requests resulting from the JavaScript interpretation will be more likely to fall between the two delimiters in this case. As before, the proxy logs and forwards the request at (1). At (2) the proxy receives an HTML tag of a form such as ⟨script language="javascript"⟩ and the proxy embeds the tag ⟨script language="javascript" src="http://start.js"⟩ into the page. This will cause a well-known request to be made from the browser to the proxy. The proxy intercepts this request at (3), places a START_JS delimiter in the log, and closes the connection to the browser. The browser continues parsing the subsequent HTML without making any changes to the rendering of the page. The request generated by the JavaScript code is seen by the proxy at (5), still inside the delimiters. When the ⟨/script⟩ tag is encountered by the proxy at (4), it is passed to the browser followed immediately by an embedded tag similar to the above. The browser will make another well-known request to the proxy, which at (6) will place the END_JS delimiter in the log file, close the connection and continue sending the page to the browser.

Meta Tags Another form of redirection is performed when the HTML content of a page contains a tag such as ⟨meta http-equiv="refresh" CONTENT="5; url=foo.htm"⟩. This causes browsers to wait the number of seconds indicated by the CONTENT attribute before automatically generating a request to the URL indicated by the URL attribute. The proxy cannot distinguish between a user making this request and the automatic request made by the browser, so in this instance we consider both the original page and the page to which the browser is directed to be two separate user requests. In some instances this is appropriate since the amount of delay is forced by the CONTENT attribute. Counting this in the overall page loading would artificially inflate the amount of time we measured that it took to load a page. However, if the delay is specified as 0, the META tag will effect an immediate request, which should be treated as the "Location" header field is treated. While we currently log META tags which may cause this form of redirection to take place, we do not currently count these redirections as belonging to the same request for a base page; the subsequent request is counted as a completely separate base page.

4.3.3 Proxy/Browser Interaction Issues

Caching The browser typically makes a request for a base page that includes some number of embedded objects such as frames and images. As the server responds to this request, the proxy parses the embedded objects and logs the information of interest. In post-processing, the objects are grouped together into web pages.

Most browsers provide for caching of objects. Cached objects may be saved to memory and/or disk. When a browser detects that a request is being made for an object that it has in the cache, it may use the object immediately, or it may generate a request which contains an "If-Modified-Since" header. The server determines whether the object has been modified, and responds either with the object, or with a 304 (not modified) Status Code, and no message body, causing the cached copy of the page to be used.

The case may arise where the cached copy of the base page is used without a request being generated, but a request is generated for an embedded object. This causes difficulty in determining how to group objects to form web pages.

Our solution to this problem is to have the user flush the browser's disk and memory caches when the proxy is first turned on. Then for each retrieved object, we determine whether the same object was embedded in some previously requested web page. If it was embedded in the immediately preceding page, it is assumed to belong to that page. If it was embedded in a page prior to the preceding page, it is assumed that that page was re-requested and was retrieved from the cache. In this case we use the prior information to group the current objects that may appear to be unrelated.

Flushing caches will negatively affect browser performance during the period when the caches are being re-populated. After this transient period, the effect will become negligible.

Browser Awareness of Proxy Existence In order to use the proxy, the browser must be made aware of the existence and location of the proxy. This causes browsers to behave slightly differently. Two differences which have been identified are (1) instead of generating the header line "Connection: Keep-Alive", the browser generates "Proxy-Connection: Keep-Alive" and (2) without the proxy the browser normally generates a request line with a relative GET request such as "GET /images/example.jpg", whereas with the proxy it generates a complete URL such as "http://www.example.com/images/example.jpg".

The header line generated by case (1) is removed as described in the section on "Proxy-Connection" above, so the difference in behavior has no effect. With case (2), however, if this request line is passed without modification directly to the server, in rare cases the server does not return the requested object, but returns a 403 Forbidden error. We handle this case by removing the "http://www.example.com" portion of the request. This worked without error for all servers we tested during development.

Streaming When making a request for an object, the browser opens a TCP connection to the proxy and sends the request. The proxy performs the DNS resolution on the name of the server, opens a TCP connection to the server and forwards the request. The server responds with the header and data for the object.

The proxy reads the data sent by the server and parses it to determine whether one of the line types described in the previous section must be logged before sending the data to the browser. For example, the proxy may encounter an HTML tag such as ⟨IMG SRC="foo.jpg"⟩, which will cause an **IMG** line type to be logged.

Since TCP is a byte stream, however, a tag may occur across two consecutive data reads by the proxy. If no provisions are made for reconstructing tags that occur across consecutive data reads, many such tags will go undetected. Therefore, the proxy maintains a per-object working buffer in which partially received tags are stored. When complete tags have been received, they are parsed and the proxy determines whether the tag must be logged.

4.4 Limitations

The source code for the Internet Junkbuster Proxy(TM) [3] (IJP) was the starting point for implementing our proxy. IJP provided a framework for filtering request and response headers, and allowed for insertion of code to parse the content of responses. It is a relatively lightweight proxy, provides fine-grained manipulation of header fields and provides platform independence. It also provides for relatively easily installation on a per-user basis.

The ability to parse content on the fly instead of capturing a complete web page before parsing the information we need to log appears to work extremely well. As qualitative evidence, we typically see browsers generate requests for objects while the base objects are being received, indicating that online parsing is working properly.

One limitation of the method is that it only measures network performance at the user's access point. It may be the case, however, that a page becomes useful to a user before all the data for the page has been received. Neither the proxy nor the post-processor can determine the point at which the rendering of a page becomes useful to a user. Another limitation is that it is somewhat labor-intensive for a user to install the proxy, configure the browser to use the proxy and to deliver the logs.

4.5 Evaluation

We evaluated our method using a single instance of browser and proxy, both running on a Sun Ultra 1, and logging to an NFS-mounted logfile. Anecdotally, we noted that highly popular web pages tend to be the most complicated in terms of content and organization of embedded objects. Therefore, we used some of the most popular sites in several categories as ranked by Top9.com [6] to test our method. Note that this evaluation was performed prior to making the design change described in section 4.3.3 under "Caching".

At this stage in development of our method, the most important task is to be able to properly group objects into web pages during post-processing. We visited the selected sites, then post-processed the resulting logfile. The results are shown in Table 2. The table lists the 41 sites we used for stress testing. For each site, it shows the total number of objects in addition to the initial request that were retrieved to satisfy a request for a web page. These additional objects include both embedded objects and Location indirections. 2232 objects in total were transferred to the browser for these 41 sites. 2191 of these were embedded objects or Location indirections. The third column shows the results of post-processing. The "# Correct" indicates the number of objects that were correctly identified as belonging to their base page. The "# Incorrect" indicates the number of objects where the method failed to identify the page to which they belonged. Objects that failed to be associated with the correct base page were identified as being base pages themselves. All the base pages

Table 2: Results of testing.

Site	# Additional Objects	# Correct/# Incorrect
www.real.com	68	68/0
www.napster.com	68	67/1
www.mp3.com	17	17/0
www.winamp.com	25	19/6
www.liquidaudio.com	51	49/2
www.peoplesound.com	95	74/21
www.audiofind.com	7	7/0
www.musicmatch.com	60	59/1
www.lyrics.com	4	3/1
www.launch.com	54	49/6
www.marsmusic.com	39	35/4
www.ubl.com	61	57/4
www.billboard.com	78	72/6
www.harmony-central.com	21	21/0
www.musiclyrics.net	45	43/2
www.music.com	68	64/4
www.mtv.com	59	54/5
www.discovery.com	54	49/5
www.abc.com	87	86/1
www.pbs.org	99	92/8
www.tvguide.com	35	32/3
www.gist.com	47	46/1
www.hollywood.com	75	72/3
www.ifilm.com	64	60/4
www.moviefone.com	62	60/2
www.movietickets.com	86	83/3
www.channel2000.com	79	76/3
www.thebostonchannel.com	91	85/6
www.imdb.com	49	49/0
www.movies.com	50	49/1
www.dragonballz.com	6	6/0
www.startrek.com	82	82/0
www.starwars.com	47	47/0
www.twistedhumor.com	18	17/1
www.funstun.com	45	45/0
www.windowsmedia.com	47	45/2
www.adobe.com	63	62/1
www.allposters.com	37	37/0
www.eonline.com	49	49/0
www.disney.com	37	35/2
www.ebay.com	62	57/5
Totals:	2191	2077/114

were correctly identified as base pages.

114 of the 2232 embedded objects, or approximately 5%, were not correctly associated with base pages. The cause of the errors were primarily related to the handling of JavaScript, although the specific reasons varied. For example, it is possible for a web page to define, within the START_JS and END_JS delimiters, a JavaScript function which will retrieve an object, but to invoke the function outside of the delimiters. During logfile post-processing, without using other heuristics, we will not detect the subsequent object as being embedded in the page that invoked the function.

The objects that were identified as base pages included both those that were correctly and incorrectly identified as base pages. There are a total of $114 + 41 = 155$ such objects.

As noted in previous sections, we plan to evaluate the degree to which the proxy affects performance as experienced by the user. The major topics for this evaluation are (1) the impact of limiting the “Accept-Encoding”, (2) the impact of disallowing “Keep-Alive” connections, and (3) the delay introduced by the proxy.

We conclude that this method is successful in logging fine-grained information about the process of downloading web pages from each user’s vantage point. The method overcomes many obstacles to gathering this data. Our evaluation demonstrates that this method is fairly accurate given the number of heuristics that must be used during post-processing. The primary limitations of this method lie in the lack of incentive for users to participate in measurement studies, and in the technical ability required to installing the software. In later chapters we will discuss other network measurement methods that address these limitations.

CHAPTER V

MEASURING DNS

Results from our previous work and other studies suggest that DNS can at times add a noticeable delay to Web page retrievals. Therefore we are interested in examining user-perceived DNS performance separately from Web performance. In this chapter we discuss the challenges involved in directly measuring DNS performance in multiple locations, and our proposed solution.

5.1 Introduction

The Domain Name System (DNS) is primarily used to map the human-readable domain names of remote machines to their IP addresses. This resolution, called a domain name *lookup*, is typically the initial step in communication between two IP endpoints when the remote IP address is not known. Thus, DNS is a critical component of the operation of many Internet applications. As described in Section 3.1, when a user indicates to a browser the URL of the desired Web page it typically contains the domain name of the target of the request. This domain name requires a DNS lookup to be performed before the HTTP request can be sent. Furthermore, a Web page may contain embedded objects — for example, images residing on different servers — that may require separate DNS lookups.

There are some studies that specifically target DNS performance [22, 28, 14]. However, many of these studies are old, or consist of performance measurements taken from a very limited number of locations — only one or two — in the Internet topology, or do not focus on performance from the perspective of the client. This raises the question "To what extent does DNS performance vary across Internet clients?" The answer has implications regarding the equity of Internet infrastructure services and the usefulness of DNS performance studies from a small number of vantage points.

Measuring DNS performance at multiple locations suffers from many of the difficulties outlined in Section 1. The primary difficulty is that the most common methods of taking measurements — for example, using `tcpdump` — require privilege that is hard to obtain in many domains.

We have three primary goals for collecting DNS performance data. First, we should capture fine-grained information about the operation of the DNS system. Such detail provides us with great flexibility in analyzing DNS performance. Second, we should collect data in such a way as to make comparison among sites as meaningful as possible. Limiting the kind of data collected at each site and controlling the method by which it is collected reduces the error in our comparisons. Finally, we should be able to collect data at multiple locations. The more locations in which we collect data, the stronger our statements are about global DNS performance.

5.2 Method

To satisfy these goals, we created a tool to run independently at multiple data collection sites to actively perform measurements. The primary component of the tool is the `named` name server[1], modified to log each event that advances the server towards resolution of the names, with a timestamp on each event line in the log. We post-process the logs for subsequent analysis. The events logged are as follows: receipt of a request to resolve a name; the sending of a request to remote servers; the receipt of responses from remote servers; the answer sent to the querying client; the removal of queries from an internal queue of pending queries; the identification of an entry in the local cache; and the identification by the server of the type of the response. We then package the modified name server with a script to drive the name lookups (the client application), a list of names to be resolved, and configuration files that allow the tool to be run by a non-privileged user at a specific port. The script utilizes the `dig` command, which invokes the `gethostbyname()` library function. This causes the client script to issue a request directly to the modified server at the server's port and to wait for a response. After five seconds if an answer has not yet been

[1]`named`, `dig` and the `gethostbyname()` library routine are all provided with the Berkeley Internet Name Domain (BIND) software distribution.

received the resolver times out and repeats the request. The resolver returns immediately after the second request has been sent, but the server continues attempting to resolve the name via retries for tens of seconds. For this reason events for different name lookups may be interleaved in the logs, requiring special care during log processing.

5.2.1 Measurement Locations

The data was collected in three groups: on NIMI [36] nodes, by colleagues with accounts on remote machines, and by members of the Linux user community who were willing to participate in this study[2]. We obtained measurements from 75 different Internet locations in 21 countries and territories[3]. Data from an additional seven sites were discarded due to anomalies in the collection process. For example, at some sites the connection was broken for some interval of time during the collection period because the individual had exceeded their maximum login time.

While we did not systematically collect specific information about the kind of Internet connection at every site where measurements were taken, many of the participants freely offered this information. From this we know that the measurement locations represent a wide variety of connection technologies, including DSL, PPP, cable modem, gigabit ethernet, etc. The timestamps in the logs show that data was collected on different days of the week, and at various times of day at each site. The dates of collection fell into two primary periods: January 2002 and late March/early April 2002.

It is unlikely that participating clients interfered with each others' measurements. Queries from our server to remote servers do not (as is normal) have the recursion desired flag set, so the remote servers should not retrieve the result in order to cache it. In Section 5.2.3 we argue that the increased load on the system has a minimal affect on performance for other participants.

[2]We emailed requests directly to individuals who identified themselves as contact persons for Linux User Groups worldwide.

[3]Countries and territories represented in our data set are: Argentina, Australia, Brazil, Costa Rica, Czech Republic, Denmark, France, Germany, Greece, Italy, Japan, Northern Ireland, Norway, Poland, Russia, Slovak Republic, South Korea, Spain, Sweden, Switzerland, United States.

5.2.2 Domain Name Sample

To perform measurements on DNS performance we first collected a large number of domain names (around 100,000). The names were collected by crawling the Web with the *Larbin* crawler [8]. Given a seed page, this crawler fetches and parses it, then recursively follows links on the resulting pages. To branch to as many web sites as possible while minimizing the impact on network and server performance, we used the default configuration of following links five deep into a web site with 60 seconds between successive requests to the same server, and reduced the number of parallel connections from 200 to 10.

The set of web sites reached by crawling the Web is highly sensitive to the starting point for the crawler [31], so we crawled multiple times, each time seeding the crawler with a different starting point drawn from a set of pages that differ in several parameters. The set of pages we chose to seed each crawl represented different values of variables such as popularity, type of web site, country of origin and type of hosting organization. The pages we used, along with the characteristics we chose to vary, are listed in Table 3.

Table 3: Starting points for crawler collecting domain names.

Seed site	Popularity	Type of site	Host organization type	US/Non-US US/Non-US
www.cnn.com	High	News	Commercial	US
www.gatech.edu	Medium	Information	Educational	US
www.parismatch.tm.fr	Medium	Entertainment	Commercial	Non-US
www.house.gov	Medium	Information	Government	US
www.chimfunshi.org.za	Low	Information	Non-Profit	Non-US
www.sina.com.cn	High	News	Commercial	Non-US
hptdc.nic.in	Medium	Information	Government	Non-US
8ball.federated.com	Low	Entertainment	Private	US

Many of the names we collected at this stage were not valid domain names. This appears to be a consequence of many factors such as mistyped links and incorrect HTML syntax. We first removed all ill-formed names. We then restricted the names to valid top-level domains. However, some invalid, yet well-formed, names remained in the name sample. Since our goal was to measure DNS performance for non-cached names, we selected a set of

Table 4: Top level domains in the domain name sample.

TLD category	Percentage of names in category
com	50%
org	14%
net	9%
edu	6%
de	3%
ru	2%
fr	1%
ca	1%
gov	1%
it	1%
151 others	less than 1% each

names that were unique up to the second level. For example, if there were two names from the example.com domain such as a.b.example.com and c.example.com, we selected only one of these names for inclusion in the final set of sample names. We did not, however, want to completely remove the effects of caching, since the normal operation of a DNS server typically has useful information cached even when the target name itself is not cached. Such information may be gTLD or ccTLD servers, or remote servers that are "closer" to the domain name being resolved. Thus, we did not force the server to flush the cache after each name lookup.

The final data set consisted of 14,983 names, each representing a unique second-level domain. The resulting names fell into the top-level domain categories in the percentages shown in Table 4.

5.2.3 Network Impact

When using active probing to perform measurements, we must quantify the impact on the system in terms of resource usage. Typically the collection takes about 4-6 hours of continuous operation to complete on each client. It requires roughly 40K outgoing packets of 40 bytes each and roughly 40K incoming packets of 300 bytes each, spread out over the entire run. On average this consumes about 700 bps outgoing and 5Kbps incoming at the measurement site.

An upper bound on the worst-case increase in root server load is calculated as the ratio X/Y, where X is the maximum number of queries across all clients to any single root server during a collection run and Y is the minimum amount of time across all clients for any collection period in seconds. The resulting worst-case increase in root server load is 832 packets/15641 seconds, or .053 packets/sec. Similarly, an upper bound on the worst-case increase in gTLD server load is 8868 packets/15641 seconds, or .57 packets/sec. These upper bounds are low enough to make certain that our measuments neither place a burden on the DNS system, nor perturb our measurements, even when multiple measurements are being taken simultaneously.

5.3 Results

In this section we examine several metrics and analyze the degree of variation in these metrics observed across the 75 measurement sites where our data collection tool was run. The primary metrics we investigate are the completion and success rates of resolving names; the mean response time for completed lookups; the root and gTLD servers that are favored by the sites; the observed fraction of names that are aliases; and the distribution of TTLs across names.

5.3.1 Completion and Success Rates

We first examine the rate of completion and the rate of success of each participating site. The response codes sent by our modified server to the client script consisted of the following values: 0, indicating no error occurred; 2, indicating a remote server failure; and 3, indicating the name does not exist. We consider a resolution to be *complete* if our server returned an answer with a response code of either 0 or 3. We consider a resolution to be *successful* if our server returned an answer with a response code of 0.

Figure 8 plots, for each site, the percentage of lookups that completed and the number that were successful. The range of values for completed lookups is [14500,14700], or 96.4% to 98.1% of the lookups. The number of successful lookups is in the range [13900,14200], or 92.7% to 94.7% of the lookups. This high number of successes can be attributed to the method by which we obtained domain names, and the filtering of invalid addresses.

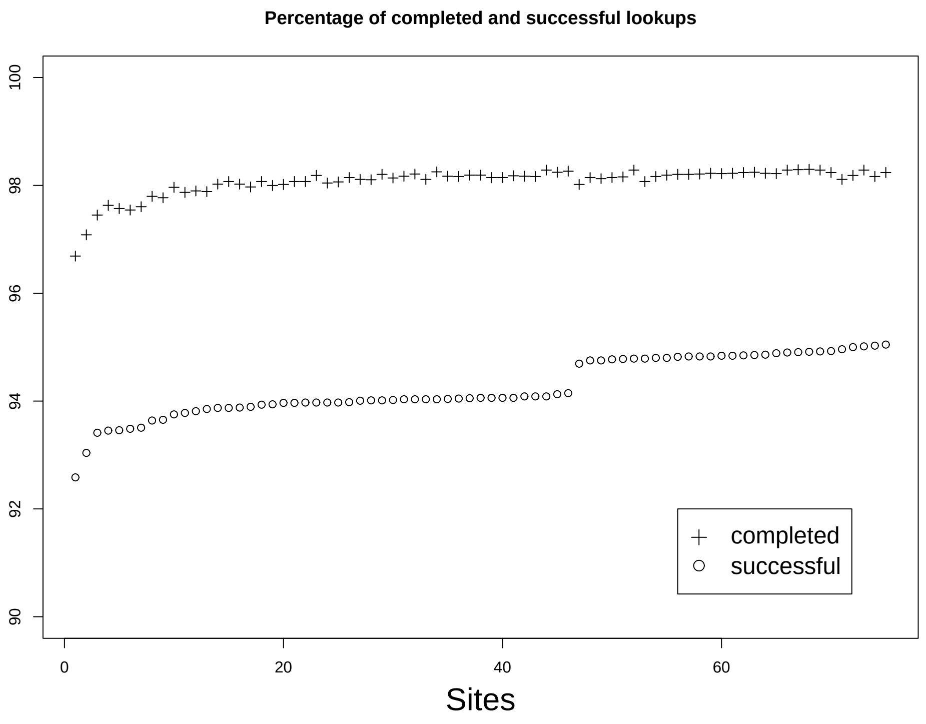


Figure 8: Percentages of completed and successful lookups.

Approximately 3% of the lookups did not complete. This can be caused by factors such as unavailable nameservers, incorrectly configured nameservers and a lack of a route to nameservers. We do not have sufficient information to quantify these problems.

In Figure 8 the sites are ordered by the number of successful lookups, exposing an interesting phenomenon. There are two weak clusters: around 14,100 successful lookups for sites 3 through 46, and around 14,200 successful lookups for sites 47 through 75. Two sites have slighly lower numbers of completed and successful lookups and do not belong to either cluster. Examining the logs for these two sites, we noted that they experienced higher numbers of retries during portions of the data collection, lasting from 3 to 16 minutes. This is likely caused by the presence of congestion close to the collection point, causing the slightly lower number of completed lookups. Other logs also seem to have experienced short periods of congestion that caused higher numbers of retries, but the congestion was not so severe as to cause more lookups to fail.

Examining the sites for each of the clusters we note that they are grouped according to

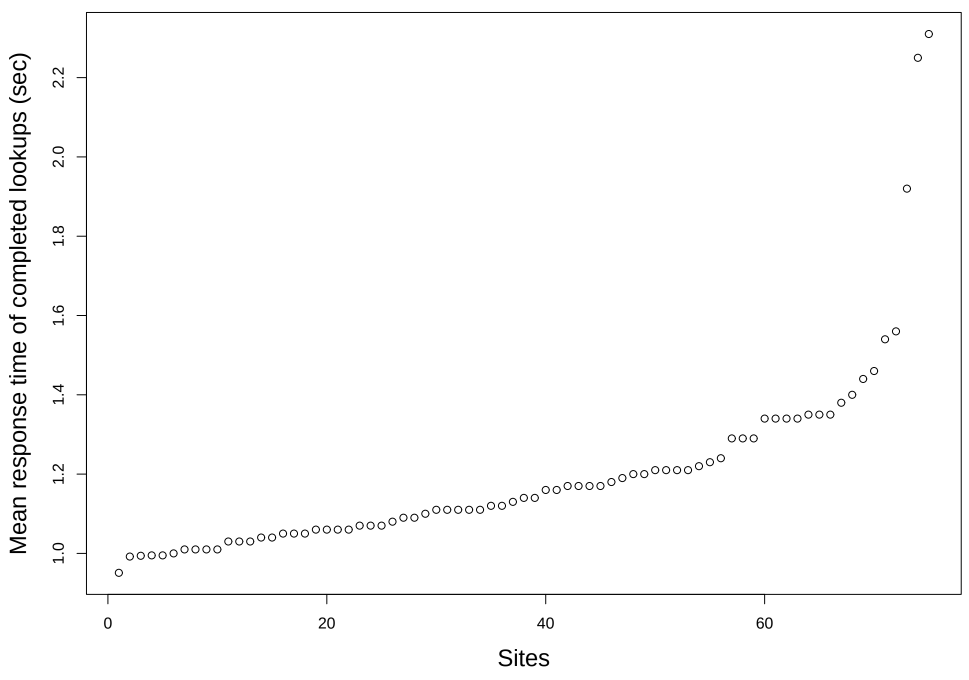


Figure 9: Mean response times for completed lookups at each site.

the dates of data collection. The data with the slightly higher number of successful lookups were all collected in January 2002 and those with a lower number of successful lookups were all collected in late March/early April 2002, showing that roughly 0.6% of the names became invalid over the course of about two months. We conclude that the numbers of completed and successful lookups for a static set of names are time-sensitive and, to the degree that one site experiences congestion more than another, they may also be location-sensitive.

5.3.2 Mean Response Time

Figure 9 shows the mean response times for completed lookups (MRTc) at each site, with sites ordered by MRTc. We see a large disparity in overall performance among the sites. The minimum MRTc is 0.95 seconds and the maximum MRTc is 2.31 seconds. This is a difference of 1.36 seconds, or a factor of 2.4. This is a very noticable delay for applications such as web browsing that require DNS lookups during human interaction.

We speculate that the four major factors affecting the MRTc for a site are the site's *connectivity*, *loss rate*, *perceived performance of root and gTLD servers*, and *location in the*

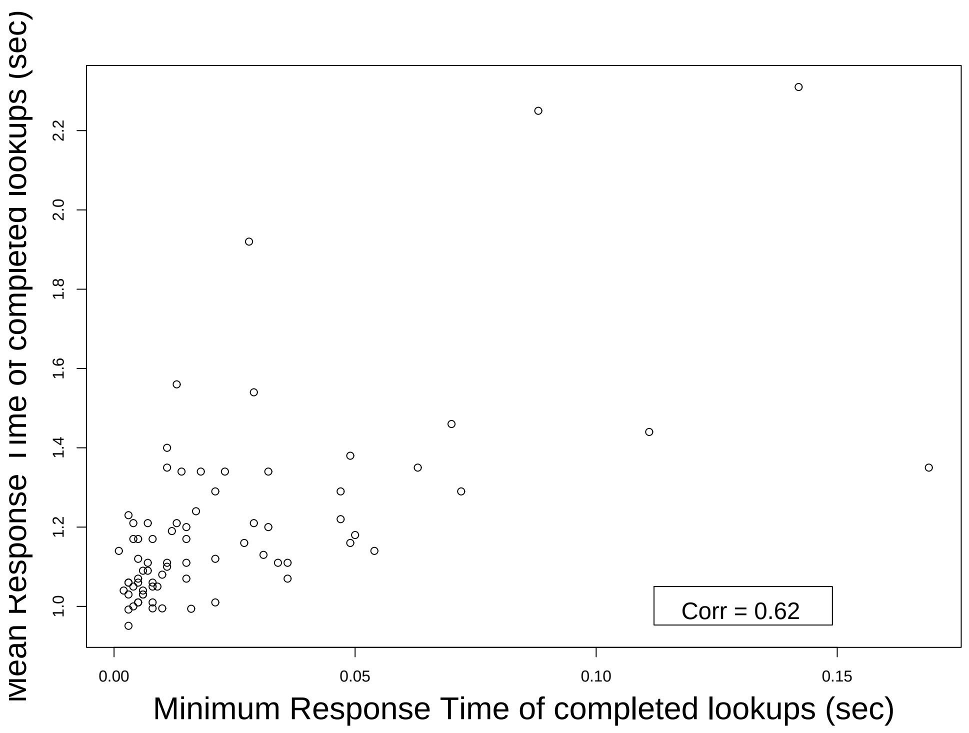


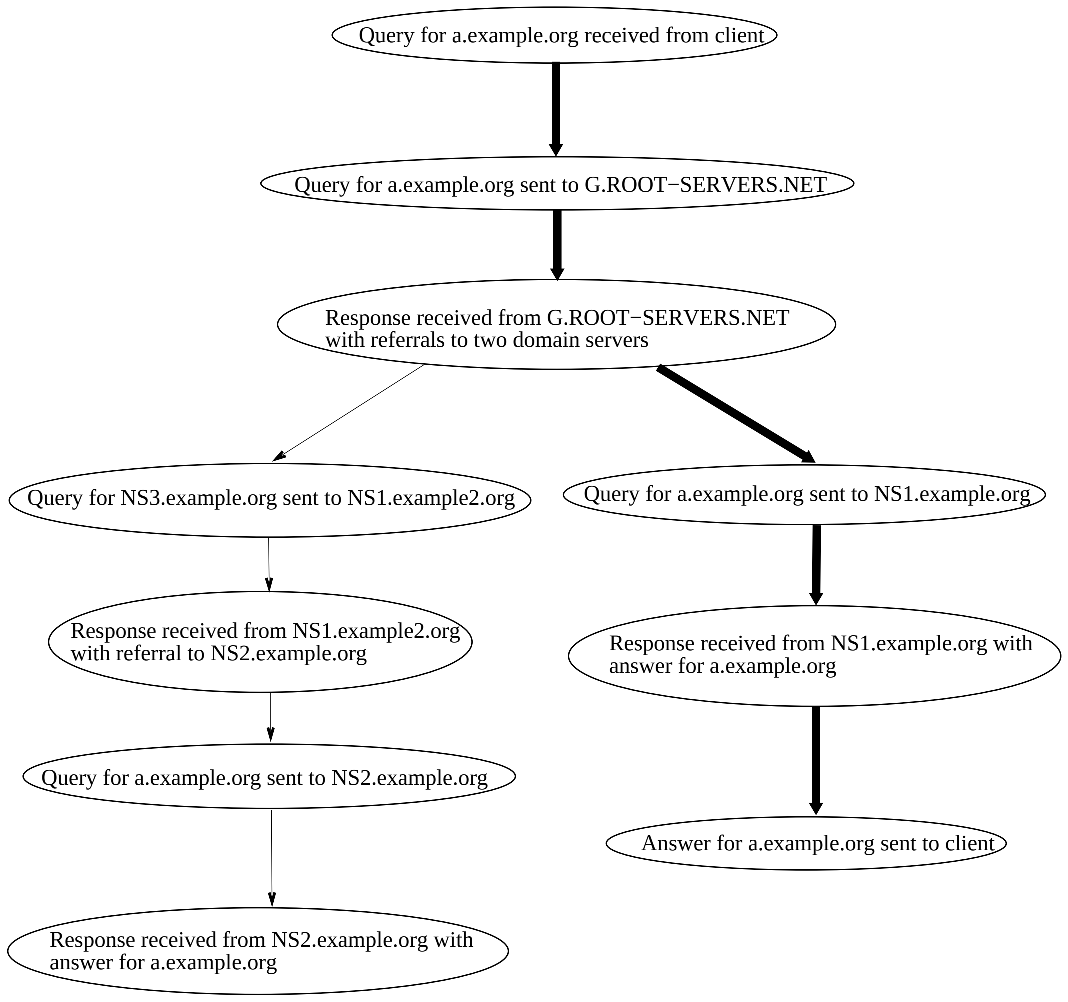
Figure 10: Minimum response times vs Mean response times for completed lookups at each site.

network relative to other name servers. We do not have fine-grained information about each of these factors for many of our sites, so we devise methods to estimate each factor from our data, and then test for correlation with the MRTc to quantify the effect of the factor on the MRTc.

5.3.2.1 Connectivity

A site's connectivity is determined by the combination of its bandwidth and its proximity to the Internet. To investigate the affects of connectivity on MRTc, we assume that the Minimum Response Time for completed lookups (MINc) is a good measure of a site's connectivity. This quantity captures the minimum round trip time for a DNS query/response to the closest name server to which a query is made. A lower MINc should correspond to a higher bandwidth connection and/or close proximity to the Internet.

In Figure 10 we plot the MRTc against the MINc at each site. Our expectation was that those sites with the highest MRTc would also have the highest MINc due to poor



Query for a.example.org received from client
Query for a.example.org sent to G.ROOT−SERVERS.NET
Response received from G.ROOT−SERVERS.NET with referrals to two domain servers
Query for NS3.example.org sent to NS1.example2.org
Query for a.example.org sent to NS1.example.org
Response received from NS1.example2.org with referral to NS2.example.org
Response received from NS1.example.org with answer for a.example.org
Query for a.example.org sent to NS2.example.org
Answer for a.example.org sent to client
Response received from NS2.example.org with answer for a.example.org

Figure 11: Example resolution tree and its critical path.

connectivity. We do see in the figure that the two sites that have the highest MRTc's (above 2.2 sec) also have higher MINc's. However, we also see two other sites that have high MINc's (above .10 sec) that also have significantly lower MRTc's (below 1.5 sec). We calculate the coefficient of correlation, ρ, of the MRTc and the MINc. As there is only a moderate correlation ($\rho = 0.62$) between the two variables, we conclude that connectivity does not sufficiently account for the higher MRTc's.

5.3.2.2 Loss Rate

The local DNS server often receives responses from remote servers that contain multiple NS (nameserver) records, or referrals, indicating other nameservers that should be contacted to resolve the name. It is quite common for the server to query multiple nameservers in parallel, leading to a “resolution tree”, where each node in the tree represents a query or response sent between machines, and each directed edge between nodes represents a causal relationship between two nodes. For example, a response from a root server may cause a query to a gTLD server. The root of the tree represents the original request, and one or more leaves may contain the A (answer) record. One such resolution tree is illustrated in Figure 11.

We use the critical path analysis technique [32] to examine the loss rate[4]. The unique path from the root of the resolution tree to the first answer sent to the client (there are sometimes multiple answers sent) comprises the *critical path* of the lookup. In practice, we determined the critical path by identifying the first answer and following the edges in the reverse direction up to the root. The nodes and edges traversed comprise the critical path for the lookup. In Figure 11, the dark arrows indicate the critical path for this resolution tree. The local server maintains a timer for outstanding queries. When a query to a remote server, or its response, are lost the timer expires and the local server resends the query to the remote server. These *retries* may also occur on the critical path for a lookup.

In Figure 12 we plot the MRTc of each site against its total number of retries along the critical path. The correlation between critical path retries and MRTc is weak ($\rho = .50$). Under the assumption that retries are a good measure of loss rate, we conclude that loss rate is not a major factor affecting lookup time for our data set. We note, however, that the loss rate varies dramatically across sites.

5.3.2.3 Root/gTLD Server Performance

We analyze the impact of the performance of root and gTLD servers by calculating the Mean Response Time for all queries sent to root servers (MRTr). Similarly we calculate the Mean Response Time for all queries sent to gTLD servers (MRTg).

Figures 13 and 14 show plots of each site’s MRTc against the site’s MRTr and MRTg, respectively. Here we see a strong correlation for each ($\rho = 0.86$ and $\rho = 0.94$), initially

[4]The idea of using critical path analysis for a portion of our work was inspired by work by Barford and Crovella [12]. They used critical path analysis to investigate various effects on the critical path profiles of TCP transactions.

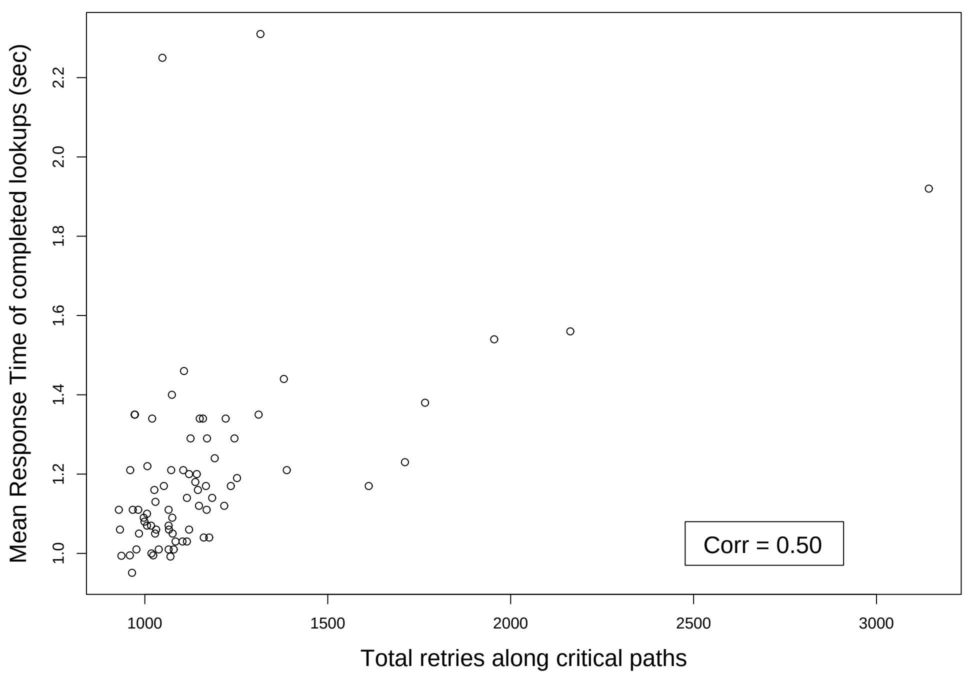


Figure 12: Total retries along critical path vs MRTc at each site.

suggesting that the performance of the root and gTLD servers have a major effect in the overall DNS performance at a site. We also see a broad range of MRTr and MRTg across the measurement sites. The range of MRTr is from 0.063 seconds to 1.41 seconds and the range of MRTg is from 0.037 seconds to 0.89 seconds.

To investigate the impact of root, gTLD and other server performance, we calculated the percentage of lookups where each of these server types was queried at some point along the critical path. The results are shown in Figure 15. We see that the percentages are quite constant at approximately 7.0% for root servers, 60.0% for gTLD servers and 98.4% for other servers. The seemingly high percentage of queries that involve root servers is caused by the fact that the root servers delegate some domains at the top level (e.g., the .se domain), and some domains at the second level (e.g., the census.gov domain). So after the initial query for the .se domain, subsequent queries for this domain will circumvent the root server, but different second-level names under the .gov domain must still go directly to the root for referral. The percentage of time spent in the critical path querying each of

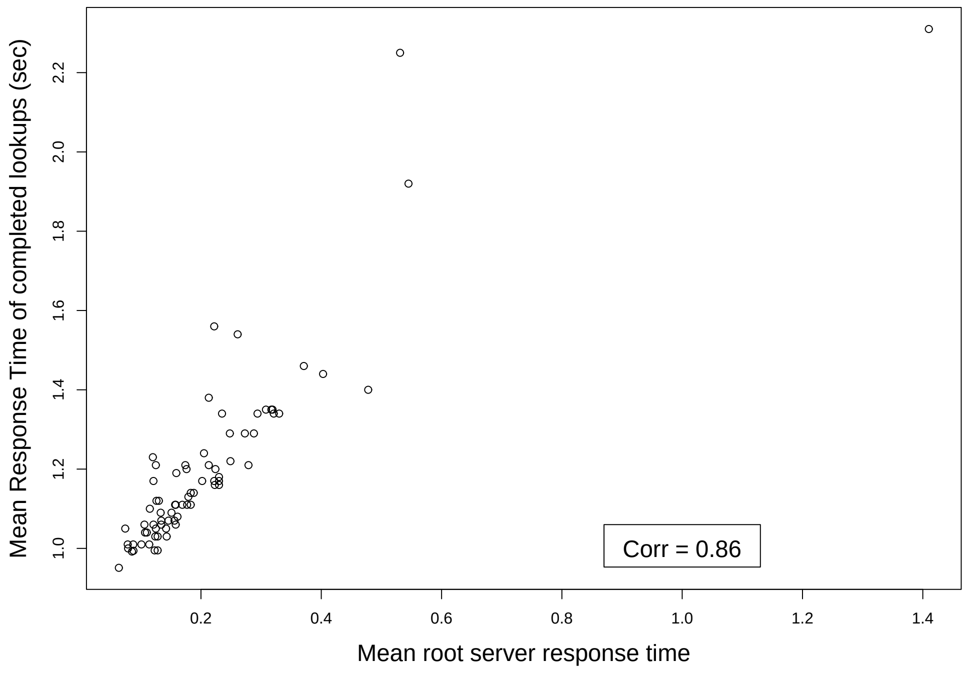


Figure 13: Mean root server response time vs MRTc at each site.

root, gTLD and other servers is shown in Figure 16.

Some implications of these results are:

- The performance of servers other than root and gTLD servers have the largest impact on performance of lookups for non-cached names. Thus, schemes that reduce this portion of the lookups, such as those employed by content delivery networks (CDNs), have the greatest impact on speeding up lookups.

- For some sites, root and gTLD server performance is quite poor, taking as much as 1.41 seconds and 0.89 seconds on average, respectively, to respond to requests. Since about 60.0% of lookups involved gTLD servers but only about 7.0% involved root servers, poor performance is not as egregious for root servers as it is for gTLD servers.

- Also because of the difference in impact on performance by root and gTLD servers, a possible service differentiator delivered by an ISP is the performance it provides from gTLD servers for non-cached names. Far less important is the performance it

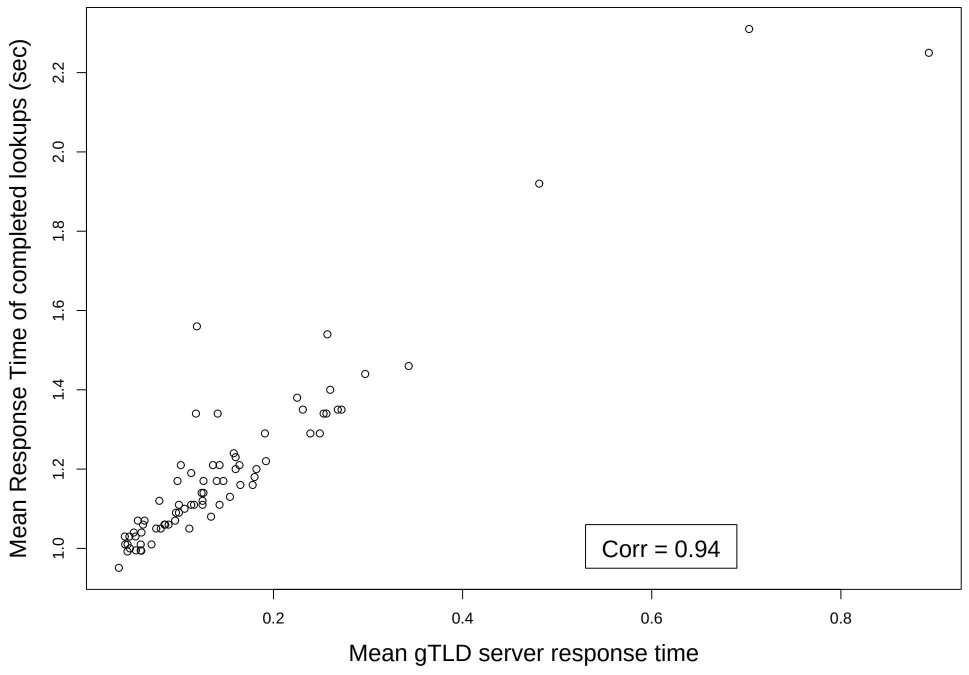


Figure 14: Mean gTLD server response time vs MRTc at each site.

provides from root servers. This could influence decisions regarding peering points and routing.

5.3.2.4 Network location relative to other servers

To estimate the location of a site relative to the rest of the Internet, we calculate the response times observed by each site to a fixed set of servers. This is related to the idea of distributed binning [37] where clients fix their location in the network based on measurements to a fixed set of servers.

We chose as our fixed set of servers the last server queried along the critical path while resolving names. We identified the set of servers that used the same set of IP addresses across all sites. We then extracted the names of the 498 servers in this set and calculated the Mean Response Time from these last servers (MRTl) for each site. In Figure 17 we plot the MRTl against the Mean Response Time for completed queries (MRTc).

The method and results of this section are similar to those of Section 5.3.2.3 where we examined the correlation between root and gTLD server performance and the MRTc

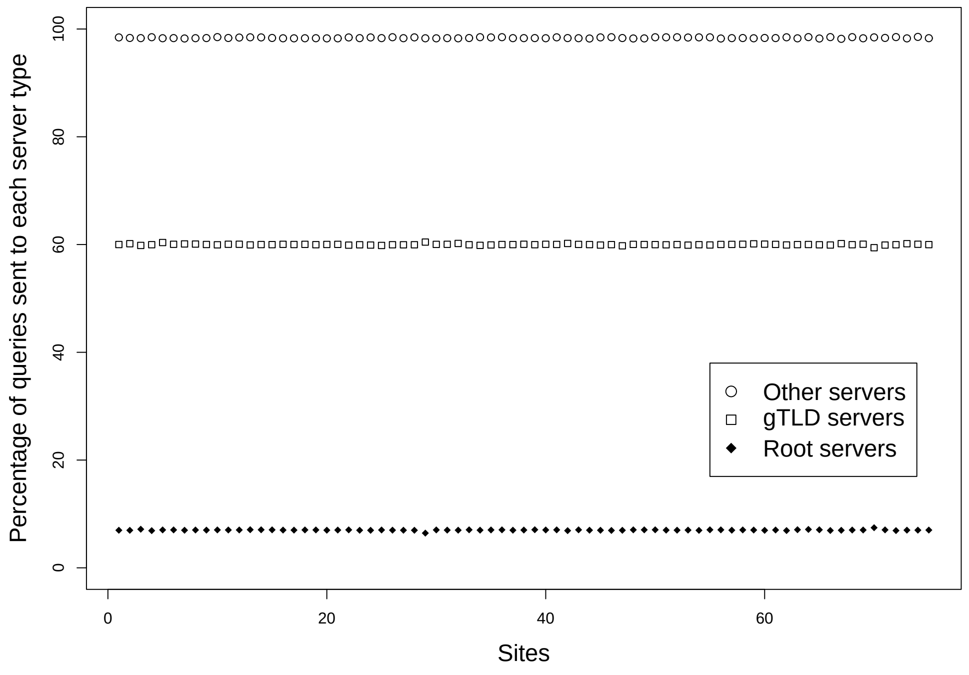


Figure 15: Percentages of queries to root, gTLD and other servers.

for each site. However, the size of those sets of servers is considerably smaller (13 of each instead of 498). From those results we only make conclusions regarding performance to those types of servers. With the larger set of servers used in this section, we can make more broad conclusions regarding distance to the rest of the Internet.

We find that the correlation between the MRTl and MRTc is strong ($\rho = 0.90$). This, under our assumption of response to the fixed set of servers indicating distance, demonstrates that the location of the site relative to the rest of the Internet is an important factor in the lookup time.

5.3.3 Root Server Interactions

The results for the root and gTLD servers in the previous section prompted us to further explore the interactions between local DNS servers and root and gTLD servers. BIND employs a server selection algorithm that seeks to minimize resolution times. The algorithm maintains a history of response times from servers when they respond to queries about a portion of the namespace. It ages this information so that all servers that will respond to

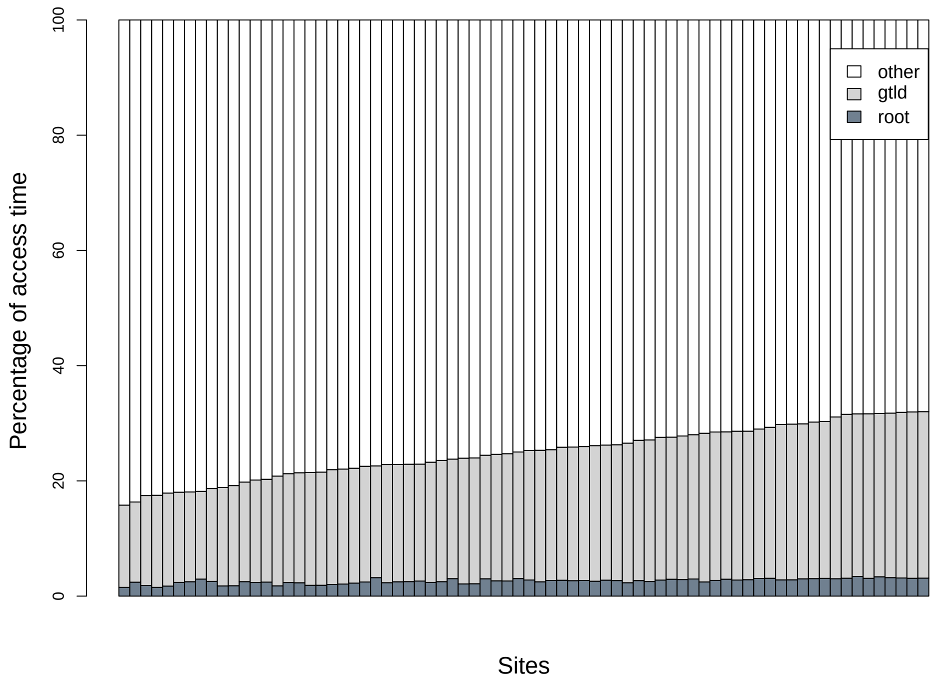


Figure 16: Percentages of time querying root, gTLD and other servers along critical path.

queries for a portion of the namespace get sampled over time. In this section we consider the degree to which queries are distributed to the root servers from each measurement site, and the response times from the root servers.

We tallied the total number of responses (not just on the critical path) received from each of the root servers by each site. We then calculated the percentage of the total from each site. We say that a site *favors* a root server if it sends greater than 10% of its root queries to that root server. Figure 18 illustrates which root servers are favored by each site. Each vertical line represents a root server and each horizontal line represents a measurement site. A dark square placed at the intersection of a site line and root server line indicates that the site favored that server. We see that four root servers ({A,D,H,I}.ROOT-SERVERS.NET in Herndon, VA, US; College Park, MD, US; Aberdeen, MD, US; and Stockholm, SE, respectively) are favored by many of the sites, whereas six root servers ({C,G,J,K,L,M}.ROOT-SERVERS.NET in Herndon, VA, US; Vienna, VA, US; Herndon, VA, US; London, UK; Marina del Rey, CA, US; and Keio, JP) are favored by few or none of

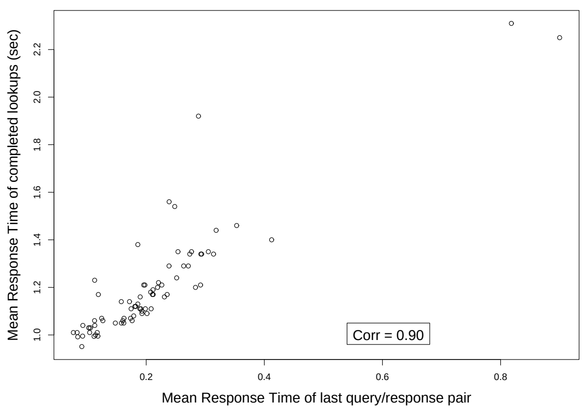


Figure 17: Mean response time for last server vs MRTc at each site.

the sites. This does not indicate that that there are not sites that would favor these servers — only that the sites where we performed our measurements did not favor these servers.

5.3.4 gTLD Servers Interaction

Figure 19 has the same form as Figure 18, but for the gTLD servers. We see that two gTLD servers ({H,I}.GTLD-SERVERS.NET in Amsterdam, NL and Stockholm,SE) are favored by many of the sites, and two gTLD servers ({J,M}.GTLD-SERVERS.NET in Tokyo, JP and Hong Kong, CN) are favored by few of the sites. Again, this does not indicate that that there are no sites that would favor these servers - only that few of the sites where we performed our measurements favored these servers.

Comparing Figure 18 and Figure 19, we see higher preferences shown for fewer root servers than we see with gTLD servers. Favoring is distributed much more evenly among the gTLD servers. The result is that there is more variation in which gTLD servers are favored from site to site than for root servers.

5.3.5 Aliases and CNAMEs

A response from a nameserver of type CNAME indicates that the query was for a domain name that is an alias for a canonical name. It is possible for the resulting canonical name to be an alias for yet another canonical name, creating a chain of aliases. Whether a name is an alias or a canonical name is configured by the domain's administrator, and is not

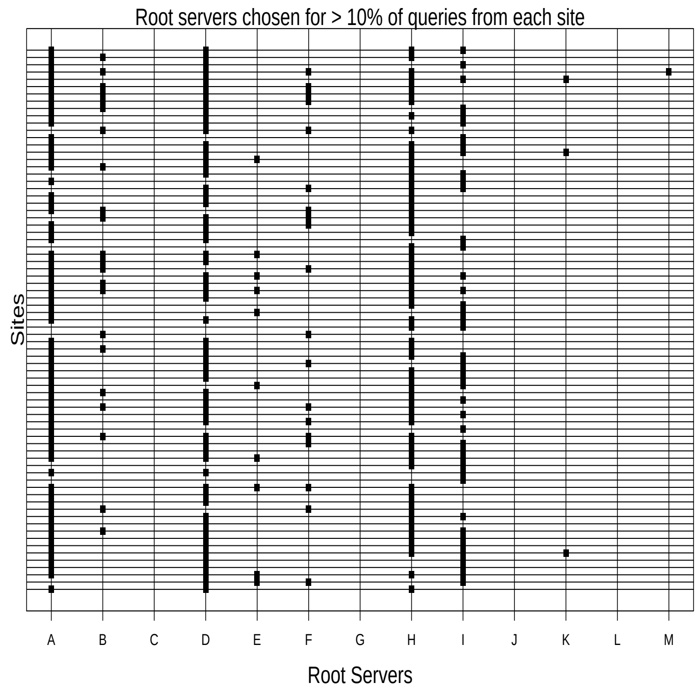


Figure 18: Root servers favored by each site.

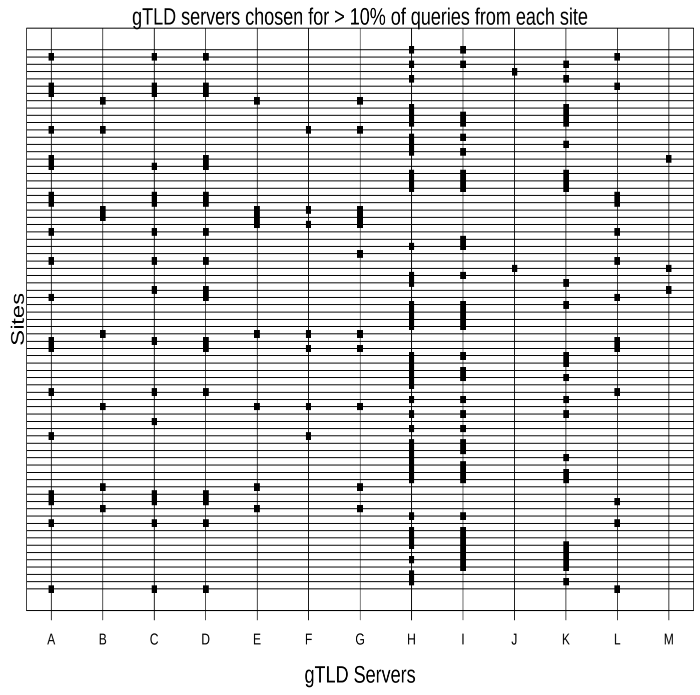


Figure 19: gTLD servers favored by each site.

Table 5: CNAME Redirections.

Number of redirections, X	Mean number (percentage) of CNAMEs with X redirections
1	3810 (96.3%)
2	138 (3.5%)
3	8.77 (0.2%)
4	1 (0.03%)

Table 6: Number of different CNAMEs per alias.

Number of different CNAME mappings, X	Number of aliases with X different mappings
1	4230 (93.6%)
2	269 (5.9%)
3	13 (0.2%)
10	1
11	1
15	1
19	1

expected to vary as a function of location. However, some CDNs leverage the function provided by CNAMEs in DNS to increase performance of web object retrieval, so we expect some portion of the actual CNAME mappings to change from site to site.

Approximately 3960, or 26%, of the names in our data set were aliases. This percentage varied only slightly across sites as expected, and this slight variation may be attributed to the variation in the number of completed lookups. Table 5 shows the mean number (percentage) of CNAME aliases in the data set. The longest chain of aliases was 4. Only 51 of the 75 measurement sites saw the chain of 4 aliases.

We also investigated the variety of CNAMEs given for aliases while performing the name lookups. Table 6 shows that by far most of the aliases resolved to the same canonical name (93.6%). Some aliases resolved to 2 and 3 different canonical names (5.9% and 0.2%, respectively), and 4 aliases resolved to 10 or more different canonical names, depending on site.

We conclude that, as expected, the number of names that are aliases is not location-sensitive, and that only a small portion of the actual CNAME mappings are location-sensitive.

5.3.6 TTLs of completed queries

Because TTLs are set by the administrator of a domain and they should be a static value for each lookup, we expect each site to show the same distribution of TTLs for the answers. We investigated this by extracting the TTLs of the names that successfully completed. We then chose bins in which to count the number of TTLs. The bins were chosen somewhat arbitrarily based on the modes of the distribution of the TTLs for one site. We then calculated the number of TTLs in each bin across all sites. From each bin we took the maximum number and the minimum number, and took the difference as the range of values in each bin. To demonstrate the degree of the difference across the sites, we first calculated the range across all sites of the number of items in each bin. We then calculated the percentage of the mean represented by this range. The result is shown in Figure 20. On the x axis is each bin. (The chosen bins are shown more precisely in Table 7.) On the y axis is the range as a percentage of the mean. We see that even with the variations in the number of names that were successfully completed at each site, the variation in the range of TTLs in each bin is extremely small. We conclude that, as expected, the distribution of TTLs seen at a site is quite constant.

We also include here the distribution of TTLs across the names. We tabulate the mean number of items in each bin in Table 7. The five most popular TTLs fall in the following bins, in decreasing order of popularity: [72001,86400], [2401,3600], [3601,7200], [38401,43200],[100001,172800].

5.4 Conclusions

This chapter presented a fine-grained study of the operation of the DNS system from multiple locations in the Internet. The goal was to compare various measures from different locations to determine which measures vary based on location and the degree to which they vary, and which measures remain relatively constant. This information will both help to

Table 7: Mean number of names in each TTL bin.

Bin end value (sec)	Mean number in bin
0	236
1	24
10	52
30	64
(one minute) 60	128
120	32
180	11
240	15
(five minutes) 300	398
360	22
600	281
900	380
1000	18
1200	108
1800	331
2400	23
(one hour) 3600	2664
7200	562
10800	331
14400	414
18000	33
21600	399
28800	369
36000	83
38400	125
43200	526
72000	144
(one day) 86400	5524
100000	54
172800	407
259200	134
345600	72
432000	37
604800	115
(ten days) 864000	18
>864000	23

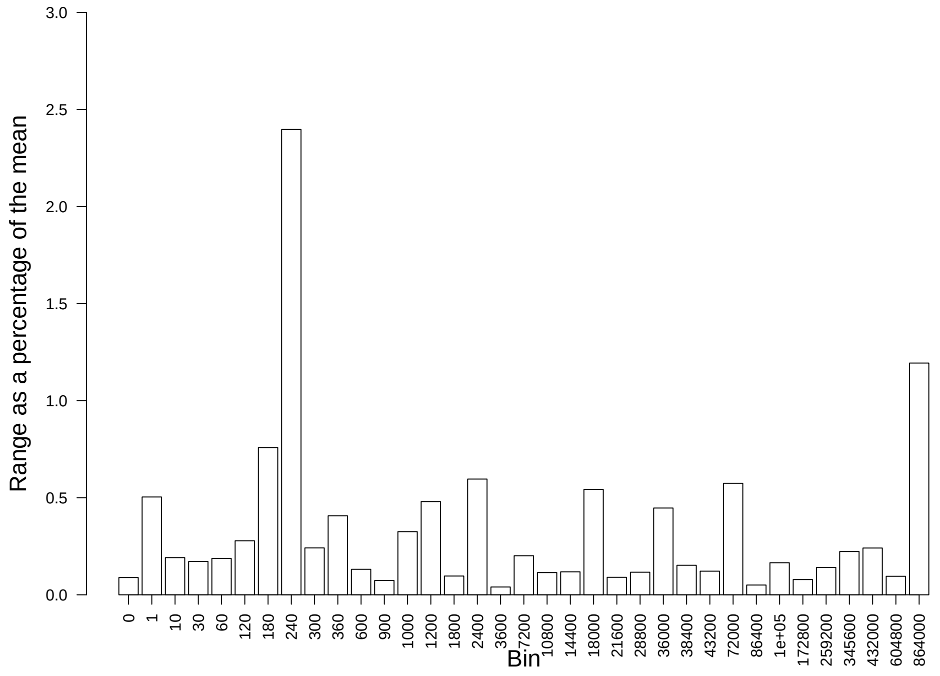


Figure 20: Ranges of number of TTLs in each bin across all sites, as a percentage of the number of TTLs in the bin.

guide engineering of the DNS and other global distributed systems, as well as guide future studies that rely on DNS performance information. We examined the correlation of DNS performance, as determined by the mean response time for completed queries, with various metrics.

Our results have demonstrated that, as expected, the DNS system tends to have low variation in those measures that are controlled by site adminstrators, like the fraction of names that are aliases and the distribution of TTLs across those names. Other measures tend to vary widely as a function of location. These include mean and median response times for completed queries and response times from root and gTLD servers.

We show that the greatest performance enhancements can be achieved by reducing the response time of servers other than root and gTLD servers. We also show that reducing the response time of gTLD servers, possibly via more equitable choice of placement of the servers, has the potential to have a very noticable impact on perceived performance. In addition, we demonstrate that root server performance has a negligible effect on perceived performance. For those measures that vary widely as a function of location, we have demonstrated that measurements from few locations may not represent the range of performance experienced across the Internet.

CHAPTER VI

WISL: AN APPLICATION FOR USER-PERCEIVED PERFORMANCE MEASUREMENT

In previous chapters we argued that the design principles and decisions guiding the development of the Internet make system-wide user-perceived measurement an elusive goal. We have also made the case that there are certain types of information that can be collected by involving the user. For example, we currently have no way to estimate the typical packet loss rate of users throughout the world. We can only sample from the point of view of the few machines that we have access to. And we have no way to adequately characterize the performance of the DNS system experienced by users in countries whose locations are remote from the root and gTLD DNS servers.

The problem caused by the lack of such data is that we can neither characterize nor verify the performance of Internet protocols and services as experienced at the end-user. We must currently rely on data collected at well-connected locations and hope that it adequately reflects the performance at hosts worldwide. However, if we can collect user-perceived performance data from actual user locations throughout the globe then we will be better able to direct our efforts in routing, provisioning and protocol tuning.

We have described some methods for overcoming the obstacles. Each method exhibits different tradeoffs in its design. The primary difficulty we face in employing these methods is that to truly achieve end-user measurements we *must* involve the end-user. This is a costly prospect for several reasons. First and foremost, the end-user must have incentive to participate in data collection. Possible incentives can be monetary remuneration or altruism. Often researchers do not have access to either the financial resources or the human resources to provide monetary remuneration. We have had some success with altruism, as have others, but this also has limitations.

In this chapter we present a new method for obtaining performance measurements from a potentially large and diverse set of locations throughout the Internet. Our method provides a completely different incentive for participation than has been employed before.

6.1 Method

We present a novel user-perceived network performance data collection tool called WISL, which stands for "**W**hat the **I**nternet **S**ounds **L**ike." The purpose of WISL is twofold. First, it provides a platform upon which to build a musical application. The music produced by WISL may take many forms, but it is all characterized by the element of chance[1]. In our case the chance input will be events that are detected as they occur in the Internet. The musical aspect of this application provides the incentive for users to download and execute the application. Second, WISL performs and reports real-time measurements of user-perceived network performance. These measurements serve as the input to the musical layer of the application, while providing a rich source of measurement data for subsequent analysis.

The architecture, described in Section 6.2, is extensible at both the music application layer and the network measurement layer. This makes it possible for researchers to provide the packages that perform different types of network measurement, and for composers to create packages that will play out their unique interpretations in sound of the current state of the network. We note that while the primary intent of WISL is to generate music, the extensible nature of WISL allows it to be employed as a network management tool as well, providing real-time audio notification of network-level events.

WISL makes possible the creation of music in a way that has never before been achieved. It allows the real-time generation of music based on events occurring simultaneously throughout the world. Part of the appeal of WISL is that it can provide the listener with an aural representation of the current state of the Internet. Composers will be attracted to the challenge of creating music that will be aesthetically pleasing while having a meaningful relationship with the state of the network. Network researchers will be drawn to creating

[1] "...something that befalls, as the result of unknown or unconsidered forces..." — Webster's Revised Unabridged Dictionary (1996). Music based on an element of chance is known as *aleatoric* music.

the modules that can report on the state of the network.

6.2 Architecture

6.2.1 Design Issues

The WISL application has a large design space. At a minimum WISL must allow measurement modules to be joined with sound playing modules. This could be accomplished by simply defining an application programming interface, or API, for both layers. However, we must balance the generality of WISL with its ease of use. That is, WISL must provide a set of services that will be useful for both researchers and composers — each of whom will be creating the packages to be used by WISL. A sufficient set of services will facilitate the participation of both composers and network researchers, relieving them of the responsibility of writing code that performs common tasks. At the same time, the architecture should as much as possible support a broad range of types of measurement that can be taken as well as types of music that can be generated. This will help to maximize participation in creating the modules required for the operation of WISL.

Ease of use by the end-user and system portability were also pervasive goals. Success in attaining these goals will aid in achieving widespread deployment of the application, thereby increasing the quality of the data collected.

Our criteria for the design of WISL take into account the three principals: the end-user, the network researcher and the composer. From the perspective of the end-user WISL should be simple to download and execute. The end-user should also be able to select among different SoundPalettes — the packages contributed by composers — since individual musical tastes vary. They should also be able to seamlessly switch between SoundPalettes in real time. Adhering to these criteria will raise the likelihood of participation by end-users.

Network researchers should be able to easily incorporate their measurement packages, called NetModules, into WISL. WISL should also facilitate the creation of NetModules by limiting the required functionality of NetModules to performing and reporting the measurements. The reporting of the measurents to the musical layer of WISL should be standardized and straighforward. For example, the music application layer of WISL may only desire notification when a measured value crosses a particular threshold rather than each time the measurement is made. WISL should alleviate the network researcher from having to manage these events by providing threshold and parameter management functionality. WISL should allow for collecting a wide range of types of measurements, and should be able to accurately timestamp events. The researcher must be able to log data for subsequent analysis. These criteria will ensure that WISL will provide the most useful platform for network researchers. They will also help to attract network researchers to contribute the NetModules that report on many different kinds of network events.

WISL should offer to the composer the ability to create a representation in sound of a wide variety of events that may occur on the network. A rich source of network event detection will make for a varied and interesting platform upon which composers can create highly varied sound environments. Furthermore, WISL should support the creation of any kind of music that can be realized within the framework of aleatoric music. Broadly, the two main categories of music that must be supported are music with a regular pulse or beat, and music without a beat. The beat-capable music criterium poses one of the most difficult technical challenges of WISL, but will allow the WISL environment to support the full range of creativity of participating composers. Finally, WISL should allow composers to create SoundPalettes without requiring them to become software engineers. They should be able to contribute sounds and define the rules that map network events to sounds with minimal effort expended in learning how to use the WISL environment.

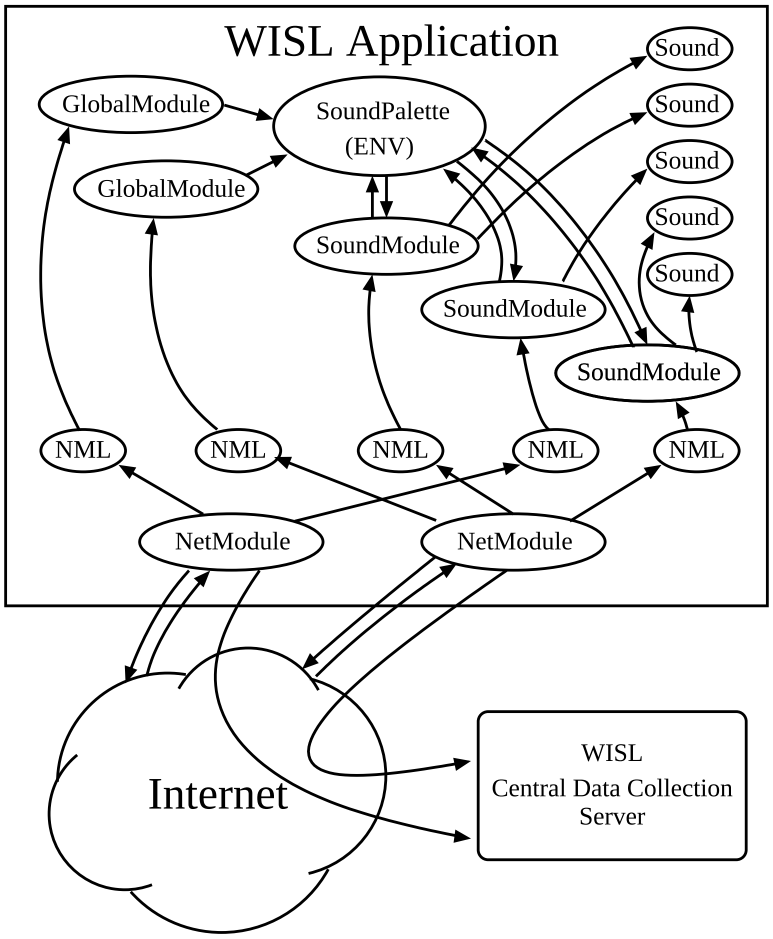


Figure 21: Simplified WISL architecture.

6.2.2 WISL Design

To meet our criteria of ease of use, extensibility, platform independence, WISL is written in java and utilizes the JavaSound API. A simplified diagram of the WISL architecture is shown in Figure 21. At the lowest layer of the application are a set of *NetModules*. Each NetModule performs a measurement task as specified by its creator. A short list of examples of the conditions that are detectable by NetModules include:

- current percentage of packet loss
- DNS response time
- sudden changes in delay or throughput
- congestion at a distant Web server

- the execution or termination of WISL at other end-user locations

WISL will instantiate at least one associated *NetModuleListener* for each active NetModule. NetModuleListeners filter and, as needed, dispatch events to the *GlobalModules* and *SoundModules* that are expecting them. GlobalModules change the current global sound environment (ENV), which determines which sets of sounds can be played out simultaneously. When a GlobalModule determines that the global environment should be changed it signals this change to the currently active SoundPalette. SoundModules receive events from the NetModuleListeners, check which global sound environment is currently active and selects the appropriate sound for playout.

The presence of NetModuleListeners eases the programming burden for NetModule creators. Each time the measurement occurs, the NetModule simply reports its measured value(s) to all NetModuleListeners that have registered their interest. The NetModuleListeners decide whether it is necessary to report these values to the active SoundPalette.

SoundModules are configured to react to events by playing particular sounds upon receipt of events from one or more NetModuleListeners. A SoundPalette consists of a set of sounds and rules that map the events to the sounds. Sounds in WISL are files that contain an encoding of a particular sound in some format. A sound may be of any reasonable length, and multiple sounds may be played simultaneously and either synchronously or asynchronously. The SoundPalette rules are specified via configuration files at application startup as described in Section 6.3. The end-user can dynamically select among the installed SoundPalettes.

JavaSound, and therefore WISL, natively supports the use of several different sound formats such as AU, WAV, and AIFF. JavaSound also supports decoders for other file formats. We have added support for the Ogg Vorbis file format to the WISL package.

Much of the internal functionality of WISL involves the following tasks:

- instantiating the necessary NetModules, NetModuleListeners, SoundModules, GlobalModules and Sounds

- passing configuration parameters to the NetModules and NetModuleListeners

- allowing NetModules to continue seamless operation while switching among SoundPalettes
- maintaining the global parameters *tempo* and *sound environment* (or musical "key")
- synchronizing sounds on a beat (if required)

6.3 Configuration

The different pieces of this application require some kind of "glue" to determine how they will interoperate. Modules may also require configuration of operating parameters. This is accomplished upon application startup via the SoundPalette Description Files (SPDFs). Each SPDF is an XML document that conforms to the WISL Document Type Definition (DTD) (Appendix A). The SPDF specifies the current operation of WISL by indicating how the events that occur on the Internet will be represented by sounds. It denotes the following configuration parameters to the application at startup:

- which NetModules must be instantiated
- operating parameters for each NetModule
- filtering parameters for the NetModuleListeners
- the sound files that will be used for playout
- expressions that describe to the SoundModules how to map events to sounds
- an initial tempo and sound environment

The SPDF, along with the set of sounds that will be played out as described in the SPDF, constitute a musical composition in WISL. Thus, the requirements for composers are that they must be able to 1) provide files that contain encodings of their desired sounds, 2) decide how the events that are reported by the NetModules will map to the sounds and 3) provide an SPDF. Requirement 1 is a common task for composers so it is reasonable to assume this ability. Requirement 2 may be aided by a descriptive README file accompanying a

NetModule. Requirement 3 is satisfied by a complete description of how to create an SPDF, with examples.

The method used for collecting data is completely at the discretion of the NetModule creator. One possibility is to run an Apache Web server at a well-known location. The initial NetModules would establish a connection with this Web server and periodically send data to the Web server, which logs the data as it is received. Java APIs exist that facilitate this scheme. Many other data collection schemes are possible.

6.4 Example NetModule: LandmarksModule

We now describe the operation of one NetModule currently being shipped with WISL. In this discussion the term *node* denotes an end-host that can provide some service to other hosts.

Consider the following problem: Given a set of nodes, Σ, connected to the Internet that deliver some service, we wish to determine with some level of accuracy which one of these, σ, is closest in the sense of latency to an arbitrary node δ. The determination of σ has applications to both peer-to-peer overlay construction and server selection [37].

LandmarksModule is being used for this measurement study in the following way. In [37] it was suggested that DNS servers could be used as landmark machines. These servers are selected because they are highly available, relatively well dispersed throughout the Internet topology and thirteen should be a sufficient number for this study. Upon WISL startup *LandmarksModule* periodically sends a single query to each of the GTLD servers and records their response times. It employs a timeout such that any query not received before the timeout expires is considered lost and is assigned a response time equal to the timeout.

A WISL Data Collection Server (DCS) listens for connections at a location known to LandmarksModule. LandmarksModule registers with the DCS by connecting to it and sending the measured response times to the DCS. The DCS employs some scheme to choose σ from the set of previously registered nodes with the new node acting as δ. The DCS then sends information about the chosen node, σ, along with a random set of other nodes, to δ.

δ then measures its latency to each of the nodes in this set to collect the round trip times and sends them all back to the DCS. The DCS periodically sends δ an update of the total number of nodes currently running, along with an update of the node chosen that is closest to δ.

There are many choices for events that can be generated by LandmarksModule and reported to the SoundPalettes for playout. We chose the following events:

- **numnodes:** the number of nodes currently running LandmarksModule
- **numneighbors:** the number of nodes deemed to be neighbors of the current node, δ
- **landmarksperf:** the minimum response time in milliseconds for all DNS queries sent out this period
- **neighbormin:** latency to the closest neighbor, σ
- **neighborfail:** failure of an attempt to communicate with a neighbor
- **percentagelost:** percentage of failed communications with neighbors
- **landmarksping:** receipt of a packet from a neighbor

Like all WISL NetModules, *LandmarksModule* is written in Java by extending the abstract NetModule class and implementing the *parseParameters()* and *beginMeasurement()* methods. These methods are required and are invoked by other WISL modules. The NetModule reports events to its NetModuleListeners by invoking the *fireEvent()* method of the NetModuleEvent class. The NetModule creator provides an accompanying file for the composer that describes the operation of the NetModule, the configuration parameters it accepts and the two events it reports.

We have also created a SoundPalette, called "Landmarks Demo" that utilizes the events generated by two NetModules, *LandmarksModule* and *PeriodicModule*. PeriodicModule generates an event called *periodic* every configurable period of time. In the case of Landmarks Demo the period is set to 20 seconds.

We require the *latency* measurement to be reported once every two seconds, and the *participants* measurement is configured to occur once every thirty seconds. Arbitrary users download and execute WISL and select the Landmarks Demo SoundPalette. When they do so, the *LandmarksModule* measurements, event reporting and data collection begin. The music representing the composer's interpretation of the events is generated on the user's machine until the user chooses another SoundPalette or closes the application.

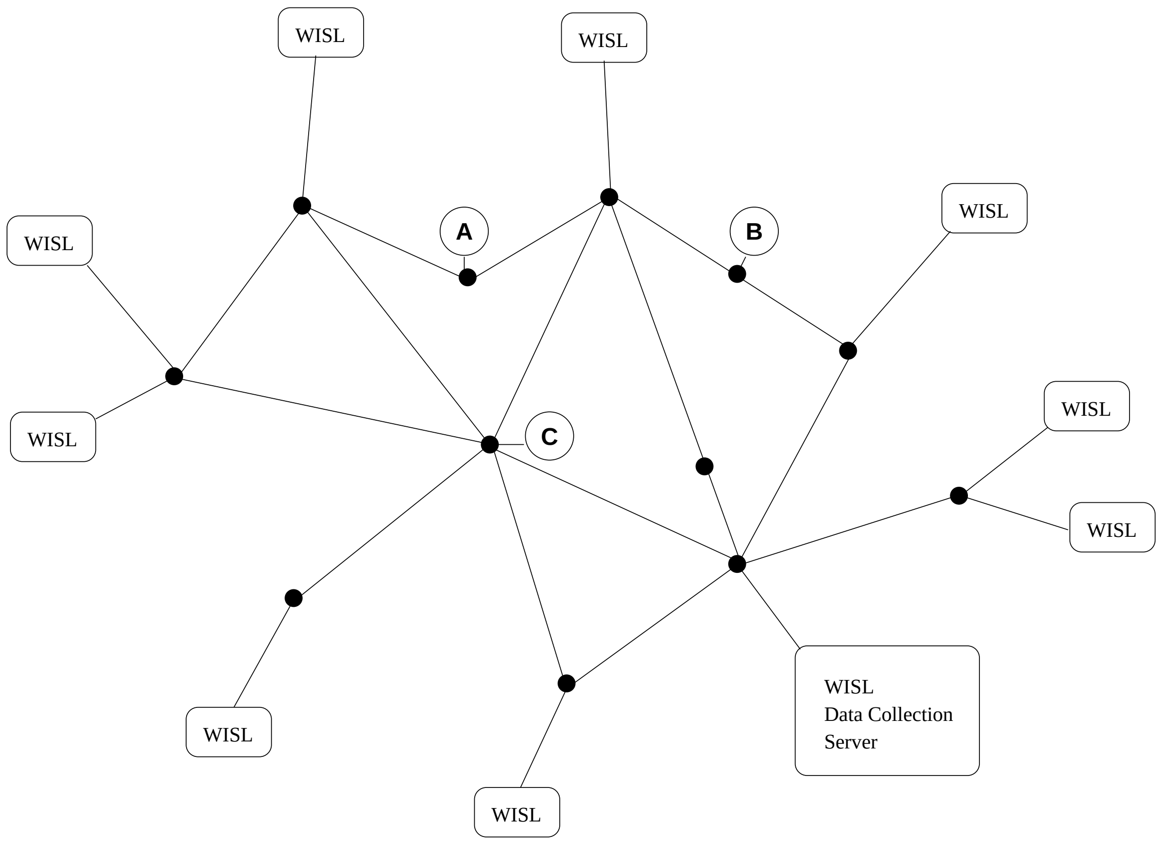


Figure 22: Multiple instances of WISL performing landmark measurements.

Figure 22 is a depiction of nine separate instances of WISL executing simultaneously at nine different end-user locations. (In this figure only three of the thirteen GTLD servers are shown.) While the DCS is collecting the data and performing its calculations it logs the information it receives. The data can then be used for subsequent analysis. In this example, different algorithms for choosing the closest node can be tested and compared.

6.5 Privacy

Collecting data about a user's network raises privacy issues. The question that must be addressed is whether data can be collected that would make public personal data of the users that they would prefer to keep private. The answer is that it is possible with any software system that allows module plugins also makes it possible for unscrupulous people to create pernicious modules. If the user installs these modules then they are subject to whatever the creator has programmed them to do. This problem is not limited to WISL, but any similarly extensible system. With WISL we attempt to mitigate this problem in the following way. We make official NetModules available only on the WISL website. The official NetModules are those that share information only about the performance of the network services. Also, each NetModule includes a plain text description file that informs the user exactly what data is collected. If the user objects to any of the data collection, the NetModule can easily be removed. When unofficial NetModules are installed, it is the user's responsibility to ensure that data is not being collected that compromises the user's privacy.

6.6 Conclusion

The research goal of WISL is to overcome the problem of collecting network performance data at end systems. It does so by building incentive to participate into the application itself. All the functionality of WISL described above has been implemented, and WISL has been through two major software releases. The current release is **WISL 0.4**. WISL is available as an open-source software package hosted at SourceForge [5]. We are continually making refinements in the form of bug fixes and new features. There are two demo SoundPalettes included in the WISL package: "Landmarks Demo" and "Loss Rate Demo".

WISL is an interesting and novel application in its own right, providing an outlet for the imaginations of both NetModule writers and composers. To our knowledge it is the first artistic application that will react to real-time events occurring throughout the globe. The music generated by WISL will be different each time it is executed, and may sound quite different in different places.

The choice of language for WISL — Java — has pros and cons for the network researcher. Java was chosen to ease the task of making the application available on multiple platforms. Working with sound devices across multiple platforms can be very difficult and error-prone. Java in combination with JavaSound alleviates many problems, making it more likely that end-users throughout the globe will be able to make use of WISL. Also, Java simplifies the task of incorporating NetModules and SoundPalettes.

The choice of Java does impose limitations on the types of measurement that can be performed. Measurements that are sensitive to timing variations on the order of a few milliseconds may or may not be suitable for WISL. The extensible nature of WISL implies that there are many possibilities for network measurement. It is anticipated that the measurements most suitable for incorporation in the WISL environment will exhibit the following characteristics:

- focus on end-user measurement
- do not require timestamping with an accuracy on a finer timescale than a few milliseconds
- can provide useful data even without control over the exact number and location of end-systems that are participating in measurement

CHAPTER VII

CONTRIBUTIONS

There is much about Internet operation that has yet to be discovered. Collecting data that provides a more complete picture of the impact of Internet behavior on the end user is an elusive, yet important goal. This thesis makes major contributions in measuring Internet performance as experienced by the end user. It describes several different methods of data collection and considers the tradeoffs exhibited by each method. The contributions of each method are summarized in Sections 7.0.1, 7.0.2 and 7.0.3.

7.0.1 Using a Proxy to Measure User-Perceived Web Performance

Much of Web performance research has focused on caching, TCP performance, network congestion and Web server performance. Our contribution to this field is the development of a method and a tool to collect Web performance data at the end user. In the process we described many practical issues that we encountered and our solutions to overcome those problems.

We collected data from users in different locations and provide an evaluation of the accuracy of our method. We provide a discussion of the tradeoffs involved in using this method for collecting user-perceived performance data. We find that while our method is accurate, the primary difficulty in collecting data on a wide scale is in the amount of configuration required by the users. In our later work we pay special attention to the ease of user configuration.

7.0.2 Diversity in DNS Performance Measures

Results from our previous work and others suggest that DNS can at times add a noticeable delay to Web page retrievals. We observed that DNS performance studies either examined data at the root servers, or from very few locations. This led us to consider whether DNS performance varies from location to location. Demonstrating heterogeneity in DNS performance implies that there is much work left for the networking community in measuring and characterizing not only DNS performance.

To this end, we created a DNS measurement package that performed lookups on a diverse set of domain names. The package provided a fine-grained view of the process of looking up names throughout the DNS namespace. We demonstrated that DNS performance measures that are subject to network conditions can vary greatly from location to location. We also verified that DNS measures that are subject to administrative configuration tend to vary very little, although we did identify some intentional variation. Furthermore, we provide evidence that although much DNS performance work has focused on root and gTLD servers, in practice they are not the source of problems in DNS performance.

7.0.3 WISL: An Application for User-Perceived Performance Measurement

While we were successful in our earlier work in gathering data from multiple locations, we found that user incentive and ease of use are the primary obstacles to wide scale deployment of our measurement packages. To address this problem we developed a novel application whose very operation is dependent upon performing and sharing network measurements. The application — called What the Internet Sounds Like, or WISL — detects the current state of the network and plays sounds that are representative of the network state. For the end user the primary uses of WISL are as a real time music generator or as a network management tool. By providing the end user with incentive to run the application we are able to gather data from portions of the network that we were previously unable to obtain.

Since the eventual success of the application is highly dependent upon the end user's musical taste, we designed the application to be highly extensible. WISL can provide the user with many different musical "channels", or SoundPalettes, to choose from. WISL is also extensible at the network layer; new measurement modules can be easily incorporated into WISL.

APPENDIX A

SOUNDPALETTE.DTD

```
<!ELEMENT soundpalette (netmoduleinit+,globalmodule*, soundmodule+)>
<!ATTLIST soundpalette
initialtempo CDATA #REQUIRED
initialenv CDATA #REQUIRED>
<!ELEMENT netmoduleinit (param*)>
<!ATTLIST netmoduleinit
        id CDATA #REQUIRED
  instance CDATA #REQUIRED>
<!ELEMENT globalmodule ((netmodule|operator)+)>
<!ATTLIST globalmodule
tempochange CDATA #REQUIRED
envchange CDATA #REQUIRED>
<!ELEMENT soundmodule (sound+, (netmodule|operator)+)>
<!ATTLIST soundmodule
        id CDATA #REQUIRED
type (default) #REQUIRED>
<!ELEMENT sound
        (filename, envlist, temporange, volume?, repetitions?,
         noqueue?, flushqueue?, mute?, stickymodulo?)>
<!ELEMENT filename (#PCDATA)>
<!ELEMENT envlist (#PCDATA)>
<!ELEMENT temporange (#PCDATA)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT repetitions (#PCDATA)>
```

```
<!ELEMENT noqueue EMPTY>
<!ELEMENT flushqueue EMPTY>
<!ELEMENT mute EMPTY>
<!ATTLIST mute
modules CDATA #REQUIRED>
<!ELEMENT stickymodulo (#PCDATA)>
<!ELEMENT operator ((operator|netmodule)+)>
<!ATTLIST operator
type (and|or) #REQUIRED
within CDATA #REQUIRED>
<!ELEMENT netmodule (event+)>
<!ATTLIST netmodule
        id CDATA #REQUIRED
  instance CDATA #REQUIRED>
<!ELEMENT event (param*,listener)>
<!ATTLIST event
name CDATA #REQUIRED>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param
name CDATA #REQUIRED>
<!ELEMENT listener (param*)>
```

REFERENCES

[1] "E2E piPES." http://e2epi.internet2.edu/E2EpiPEs/overview.html.

[2] "ICPLD." http://icpld.northernmost.org.

[3] "Internet Junkbuster Proxy(TM)." http://www.junkbuster.com/ijb.html.

[4] "Keynote." http://www.keynote.com/keynote_method/keynote_method_methodology.html.

[5] "SourceForge." http://sourceforge.net.

[6] "Top9.com." http://www.top9.com/.

[7] "WebPerf." http://www.webperf.org.

[8] Ailleret, S., "Larbin." http://larbin.sourceforge.net/index-eng.html.

[9] Akella, A., Seshan, S., and Shaikh, A., "An empirical evaluation of wide-area internet bottlenecks," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, pp. 101–114, ACM Press, 2003.

[10] Allman, M., Blanton, E., and Eddy, W., "A scalable system for sharing internet measurements," in *Proceedings of the Passive and Active Measurement Workshop*, (Fort Collins, Colorado), March 2002.

[11] Balakrishnan, H., Stemm, M., Seshan, S., and Katz, R. H., "Analyzing stability in wide-area network performance," in *Measurement and Modeling of Computer Systems*, pp. 2–12, 1997.

[12] Barford, P. and Crovella, M., "Critical path analysis of TCP transactions," in *Proceedings of ACM SIGCOMM*, pp. 127–138, 2000.

[13] Bernstein, D. J., "djbdns." http://cr.yp.to/djbdns/notes.html.

[14] Brownlee, N., Claffy, K., and Nemeth, E., "DNS measurements at a root server," in *Global Internet 2001*, November 2001.

[15] Clark, D. D., "The design philosophy of the DARPA internet protocols," in *SIGCOMM*, (Stanford, CA), pp. 106–114, ACM, Aug. 1988.

[16] Cohen, E. and Kaplan, H., "Prefetching the means for document transfer: A new approach for reducing web latency," in *Proceedings of the IEEE INFOCOM '00 Conference*, 2000.

[17] Cohen, E. and Kaplan, H., "Proactive caching of DNS records: Addressing a performance bottleneck," in *Proceedings of the Symposium on Applications and the Internet (SAINT)*, pp. 85–94, 2001.

[18] Cranor, C. D., Gansner, E., Krishnamurthy, B., and Spatscheck, O., "Characterizing large DNS traces using graphs," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[19] Crovella, M. and Bestavros, A., "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," in *Proceedings of SIGMETRICS'96: The ACM International Conference on Measurement and Modeling of Computer Systems.*, (Philadelphia, PA), May 1996. Also, in Performance evaluation review, May 1996, 24(1):160-169.

[20] Cunha, C., Bestavros, A., and Crovella, M., "Characteristics of World Wide Web Client-based Traces," Tech. Rep. BUCS-TR-1995-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[21] Danalis, A. and Dovrolis, C., "ANEMOS: An Autonomous NEtwork MOnitoring System," in *Proceedings of the Passive and Active Measurement Workshop*, (La Jolla, CA), April 2003.

[22] Danzig, P. B., Obraczka, K., and Kumar, A., "An analysis of wide-area name server traffic: A study of the domain name system," *Proc. of the ACM SIGCOMM '92*, January 1992.

[23] Elz, R. and Bush, R., "RFC 2181: Clarifications to the DNS specification," July 1997.

[24] Elz, R., Bush, R., Bradner, S., and Patton, M., "RFC 2182: Selection and operation of secondary DNS servers," July 1997.

[25] Grossglauser, M. and Krishnamurthy, B., "Looking for science in the art of network measurement," in *IWDC*, pp. 524–535, 2001.

[26] Gummadi, K. P., Saroiu, S., and Gribble, S. D., "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, (Marseille, France), November 2002.

[27] Hafner, K. and Lyon, M., *Where Wizards Stay Up Late : The Origins of the Internet.* Touchstone Books, 1996.

[28] Jung, J., Sit, E., Balakrishnan, H., and Morris, R., "DNS performance and the effectiveness of caching," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2001.

[29] Kalidindi, S. and Zekauskas, M. J., "Surveyor: An infrastructure for internet performance measurements," in *Proceedings of INET '99*, (San Jose, CA), June 1999.

[30] Koletsou, M. and Voelker, G., "The medusa proxy: A tool for exploring user-perceived web performance," June 2001.

[31] Krishnamurthy, B. and Rexford, J., *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement.* Addison Wesley, 2001.

[32] Lockyer, K., *An Introduction to Critical Path Analysis.* New York:Pitman Publishing Company, 1964.

[33] Mockapetris, P. V., "RFC 1034: Domain names — concepts and facilities," Nov. 1987.

[34] Mockapetris, P. V., "RFC 1035: Domain names — implementation and specification," Nov. 1987.

[35] Paxson, V., Adams, A., and Mathis, M., "Experiences with NIMI," 2000.

[36] Paxson, V., Mahdavi, J., Adams, A., and Mathis, M., "Creating a scalable architecture for internet measurement," *IEEE Communications*, vol. 36, pp. 48–54, August 1998.

[37] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S., "Topologically-aware overly construction and server selection," in *Proceedings of the IEEE INFOCOM '02 Conference*, 2002.

[38] Shaikh, A., Tewari, R., and Agrawal, M., "On the effectiveness of DNS-based server selection," in *Proceedings of IEEE Infocom*, April 2001.

[39] Simpson, C. R. and Riley, G. F., "NETI@home@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements," in *Proceedings of the Passive and Active Measurement Workshop*, (Antibes Juan-les-Pins, France), April 2004.

[40] Spring, N., Wetherall, D., and Anderson, T., "Scriptroute: A public internet measurement facility," in *USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[41] Tierney, B. L., Gunter, D., Lee, J., Stoufer, M., and Evans, J. B., "Enabling network-aware applications," in *HPDC-10*, August 2001.

[42] Wills, C. and Shang, H., "The contribution of DNS lookup costs to web object retrieval," Tech. Rep. TR-00-12, Worcester Polytechnic Institute, July 2000.

[43] Wills, C. E., Mikhailov, M., and Shang, H., "Inferring relative popularity of internet applications by actively querying DNS caches," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, October 2003.