Application-Layer Anycasting

Samrat Bhattacharjee
Mostafa H. Ammar
Ellen W. Zegura (Corresponding Author)
Viren Shah
Zongming Fei

Networking and Telecommunications Group
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{bobby,ammar,ewz,viren,fei}@cc.gatech.edu
404-894-1403
404-894-0272 (fax)

Abstract

Server replication is a key approach for maintaining user-perceived quality of service within a geographically wide-spread network. The anycasting communication paradigm is designed to support server replication by allowing applications to easily select and communicate with the "best" server, according to some performance or policy criteria, in a group of content-equivalent servers. We examine the definition and support of the anycasting paradigm at the application layer, providing a service that maps anycast domain names into one or more IP addresses using anycast resolvers. In addition to being independent from network-layer support, our definition includes the notion of filters, functions that are applied to groups of addresses to affect the selection process. We consider both metric-based filters (e.g., server response time) and policy-based filters; we further allow filtering both at the anycast resolver and local to the anycast client. A key input to the filtering process is metric information describing the relative performance of replicated servers. We examine the use of various techniques for maintaining this information at anycast resolvers.

1 Introduction

The Internet is increasingly being viewed as providing services, and not just connectivity. As this view becomes more prevalent, it becomes important to provide, within the Internet, explicit support for the efficient delivery of networked services. An important consideration in the provision of such services is the ability of a service to meet the demands of a large number of users that are geographically wide-spread. It is also important that the user-perceived quality of service (e.g., response time, throughput, reliability) be maintained at an acceptable and competitive (in the case of commercial services) level. This is often referred to as the *scalability* of the service.

There have been several approaches proposed for improving the scalability of a networked service. These include server replication [1], caching [2, 3], batching of requests at the server [4] and multicasting of server responses over the network [5, 6]. In this paper we consider the server replication technique. In particular, we investigate the *anycasting* communication paradigm which has been proposed [7] to support server replication.

As originally defined [7], anycasting provides:

> "a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address."

In this definition, an *IP anycast address* is used to define a group of servers that provide the same service. A sender desiring to communicate with only one of the servers sends datagrams with the IP anycast address in the destination address field. The datagram is then routed using anycast-aware routers to at least one of the servers (presumably the "best" according to some criteria) identified by the anycast address.

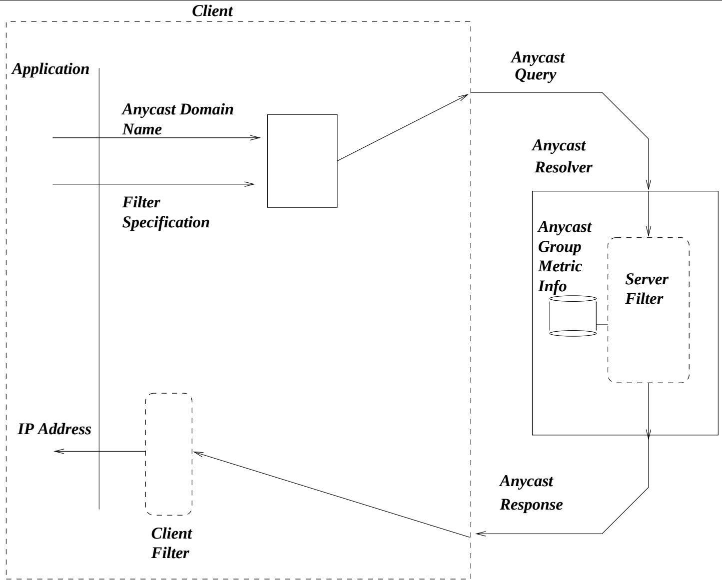
In our work we adopt a more general view of anycasting as a *communication paradigm* that is analogous to the broadcast and multicast communication paradigms. In particular, we differentiate between the anycasting *service definition* and the *protocol layer* providing the anycasting service [1]. The original anycasting proposal [7] can, therefore, be viewed as providing the anycasting service definition *and* examining the provision of this service within the IP layer.

In this paper we examine the definition and support of the anycasting paradigm at the *application layer*. Our motivation derives from the fact that network-layer-supported anycasting has the following limitations:

- Some part of the IP address space must be allocated to anycast addresses. For IPv4, several ways to allocate anycast addresses have been suggested [7].These include designating

[1]For example multicasting as a communication paradigm represents a desire to send the same message to a group of receivers. The multicast paradigm can be supported using multicast routing at the network layer or it can be supported above a unicast-only network or transport layer by using multiple unicasts. Indeed this latter approach was how many multicast problems were addressed in early system designs (e.g., the original ISIS distributed system [8]). There is clearly a distinction between multicast as a communication paradigm and how this paradigm is supported.

FIGURE 1 Anycast Name Resolution Query/Response Cycle



some existing addresses as anycast (e.g., within Class C) or creating a separate class of addresses. IPv6 proposals [9] do include a specific address space allocated to anycasting.

- The use of anycast addresses requires router support. Routers must recognize anycast addresses and forward them properly. Routers must coordinate with one another to ensure that delivery is usually made to exactly one host.
- The selection of the server to which an anycast addressed datagram is sent is made entirely within the network with no option for user selection or input.
- Consistent with the stateless nature of IP, the destination is determined on a per-datagram basis. Thus, two successive datagrams sent to an anycast address may be delivered to two different hosts that serve the address. A protocol that requires all datagrams to be delivered to a single host can use anycasting for discovery of a satisfactory host, but subsequent transmissions should use standard IP to the selected host.
- Because of the network layer's ability to efficiently determine shortest paths, it is well-suited for an anycasting service based on the selection of the closest (in number of hops or other distance or delay metrics) server. An application layer approach is better suited at handling a variety of other metrics such as server throughput or turnaround time.

Whereas network-layer support hinges around the use of anycast IP addresses, our application-layer support makes use of *anycast domain names* (ADNs). The function of an application-layer anycasting service is thus to map an ADN into one or more (multicast or unicast) IP addresses. An important feature of application-layer anycasting is that it does not require modifications to network layer operations.

This paper focuses on the design of an infrastructure to provide an application-layer anycasting service. Our design centers around the use of *anycast resolvers* to perform the ADN to IP address mapping. Clients interact with the anycast resolvers according to a basic query/response cycle illustrated in Figure 1: a client generates an *anycast query*, the resolver processes the query and replies with an *anycast response*. A key feature of the system is the presence of *metric databases*, associated with each anycast resolver, containing performance data about servers. The performance data can be used in the selection of a server from a group, based on user-specified performance criteria. We consider *metric determination* techniques that can be used to maintain the anycast resolver databases. In addition, we investigate how anycast client applications may interface with the anycast resolver.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 describes in more detail the components of application-layer anycasting, providing our definitions of replicated service, anycast groups and domain names, and anycast resolver functionality. In Section 4 we elaborate on the anycast query/response cycle, discussing the design and operation of the interface to an anycast resolver. Section 5 considers a set of metrics and metric determination mechanisms to allow accurate measurements while not unduly loading the network or the resolvers. We conclude the paper in Section 6 with an assessment of the merits of application-layer anycasting and a discussion of future work.

2 Related Work

The server or resource finding problem has been the subject of much investigation for over a decade. Initially, with low to moderate server loads, the problem was how to find the desired resource over the network knowing only its name or property. Many techniques were proposed and investigated. These include: 1) the use of multicast or broadcast communication to "touch" all the locations where the resource may reside in an attempt to find it (e.g., [10, 11]) , 2) the use of various name server architectures in order to lookup the location of the resource (e.g., [12, 13, 14]) and 3) the use of caching of a resource's location (not content) at sites where the resource is frequently accessed [15]. This early work dealt with situations where there is typically a single instance of the resource. The case of a mobile resource was addressed through interesting techniques such as the use of forwarding addresses [16]. More recently, the Service Location Working Group of the IETF is considering the design of the Service Location Protocol which allows a user to specify a set of service attributes which can be bound to a server's network address in a dynamic fashion [17].

Beginning with initial services like `ftp`, `archie`, and `gopher` and culminating more recently with the World-Wide Web, the Internet has experienced a dramatic growth in the use and provision of information services. This has resulted in heavy demands being placed on servers resulting in the desire to replicate (or mirror) servers. This adds a new dimension to the server finding problem: it is now important to find the "best" server from among many content-equivalent servers. Two notable studies in this area are: 1) the original work by Partridge, Mendez and Milliken [7] proposing the idea of anycasting and discussing its network-layer support and 2) a recent study by Guyton and Schwartz [18] which addresses the problem of locating the nearest server. The latter work also presents a classification of "best"-server location schemes. The work is related to earlier work on the `Harvest` system [19] which provides a set of tools for gathering information from various servers and efficiently indexing and searching through this information. Tools for caching and replication of indices are also used in the `Harvest` system in order to improve the scalability of the service. Another related project is the SONAR network proximity service [20] in which the authors define a service which can return the closest (in hops) server from among a provided list.

By choosing to define anycasting as a communication paradigm, we deviate somewhat from the Guyton and Schwartz classification which considers the original definition of anycasting as a network-layer-supported service. Our work investigates the complete design of application-layer anycasting and not just the metric probing aspect as discussed in [18]. We also consider using a variety of metrics (not just hop-distance as provided by SONAR) in order to provide a closer match to the application requirement.

The concept of probing the network and the servers to determine performance measures is related to various tools, and systems that are used for network management purposes. Examples of these include the `traceroute` [21], and `mwatch` [22] tools and management systems that integrate such tools such as the `Fremont` system [23]. Remote measurement and monitoring of system performance has also been explored as part of the extensive work on distributed system monitoring [24, 25]. Recently, the company Timedancer Systems has been offering a service dubbed OnTime Delivery [2] which probes a web site at regular intervals and provides a report on the performance of the web site as perceived by their client.

To avoid the scalability problems inherent in probing for performance we also explore the idea of having the server "push" onto the network its own locally generated performance observations. This is related to the *Push Caching* idea [26] where servers are in charge of pushing the desired information onto remote caches and the *server push* mechanism [27] implemented in Netscape browsers.

Finally, anycasting is related to the technique used to build scalable HTTP servers [28]. In such a scheme multiple servers are clustered and appear to the outside world a single logical server. Modifications to the DNS resolution mechanisms are made to allocate the load among the servers.

3 Generalized Application-Layer Anycasting

In this section we describe in more detail the definition of application-layer anycasting as a communication paradigm. Later sections are concerned with issues relating to the realization of this paradigm.

[2] See http://www.timedancer.com/intime.html.

3.1 What Is a Replicated Service

An anycast domain name (ADN) is used to identify a particular network service that can be provided by multiple locations on the network. We say that two servers *replicate* each other if they contain equivalent content and/or functionality from an *application perspective*. It is true that servers with identical content or functionality are considered replicas of each other, though in our definition replicated servers do not necessarily have to be identical.

The Network Time Protocol (NTP) is an example of a simple service that is widely replicated, where all servers provide identical functionality. On the other hand, whereas the many web search sites (e.g., Lycos, InfoSeek, Yahoo) carry different information, they, nevertheless, can be construed as being replicated servers. Depending on the application, the web servers for CNN Interactive, Time Magazine and USA Today could constitute a set of replicated servers as well.

In some cases multiple servers are required to provide a single instance of a service. A primary example is quorum consensus applications that arise in distributed systems. Quorum consensus is a general class of synchronization protocols for distributed systems [29, 30, 31, 32]. An operation may proceed to completion only if it is granted permission from a number of coordinator nodes. If mutually exclusive execution of operations is desired, e.g., as would be required when updating strongly-consistent replicated data, then the node executing the operation needs to collect permission from a majority of the coordinator nodes. Other applications, such as the reading of strongly-consistent replicated data, may require permission from a certain number of nodes, not necessarily a majority. By including all the coordinator nodes in a multicast group with a single group address, multicast communication can be used by a quorum collector to communicate with the set of coordinator nodes.

This type of service is typically "replicated" by making the the number of coordinator nodes larger than the required quorum. This allows any *subset* of the coordinator nodes with the appropriate quorum to be able to deliver the service. This replication is done for the usual reasons: fault tolerance and load balancing.

We also extend our notion of service to include more general functionality than information content, e.g., *compute servers*. In this case the user is interested in finding a server on which to run a particular computation. The set of available hosts can then be made to form an anycast group. The performance measure of interest to select the best server would normally be CPU load. An interesting variation is where the user has a parallel program to run that requires the use of multiple hosts. This is another instance of multiple hosts providing a single instance of a service; making the anycast group a collection of pre-formed multicast groups each containing enough hosts to execute the parallel program.

3.2 Anycast Groups

An anycast domain name (ADN) uniquely identifies a (potentially dynamic) collection of IP addresses, which constitutes an *anycast group*. An anycast group can be made up entirely of unicast IP addresses or entirely of multicast IP addresses [3]. In the former case the anycast group represents a group of servers and in the latter it represents a collection of multicast groups [4]. In both cases, the anycasting service provides a mapping from the ADN representing the group to the "best" address in the collection according to some criteria.

The motivation for collecting a set of unicast IP addresses into an anycast group is straightforward and derives from the original motivation of the anycasting paradigm. Each of the IP addresses in the group represents the address of one of the replicated servers providing the service. Reaching any one of them is acceptable (as far as service content and functionality is concerned), and the anycasting service makes a decision based on performance and/or policy metric.

We allow a collection of multicast addresses to form an anycast group in order to support the class of quorum consensus applications that we described above. For these applications, it makes sense to define multicast groups made up of subsets of the set of all coordinators, assign each subset a multicast address and collect the multicast addresses into an anycast group. The application then would use the ADN associated with this group to refer to the coordination service. The anycasting implementation would then map the ADN into a multicast group with the desired quality (e.g., least average coordinator load or distance from client).

It is also possible to define an anycast group as a collection of server domain names (or aliases) [5]. In this case the anycasting service provides a mapping from an ADN into a host domain name or alias. Obtaining the IP address in this case requires an additional DNS server lookup. On the other hand, knowing the domain names that make up an anycast group may facilitate the process of selecting from among the group members as will be discussed later.

3.3 Anycast Domain Names and Resolver Architecture

The issue of the structure of anycast domain names influences the operation of the anycasting system in general, and the anycast resolver architecture in particular. We propose an approach in this paper that is derived from the Internet naming and directory service architecture. Such an approach makes it straightforward to integrate our anycasting architecture into the existing Internet infrastructure.

We prefer to view the anycast resolver as logically distinct from other name servers like DNS [12], allowing us to consider issues related to anycast resolver design separately from other name service issues. In reality, the functions of an anycast resolver could be integrated with the operation of DNS.

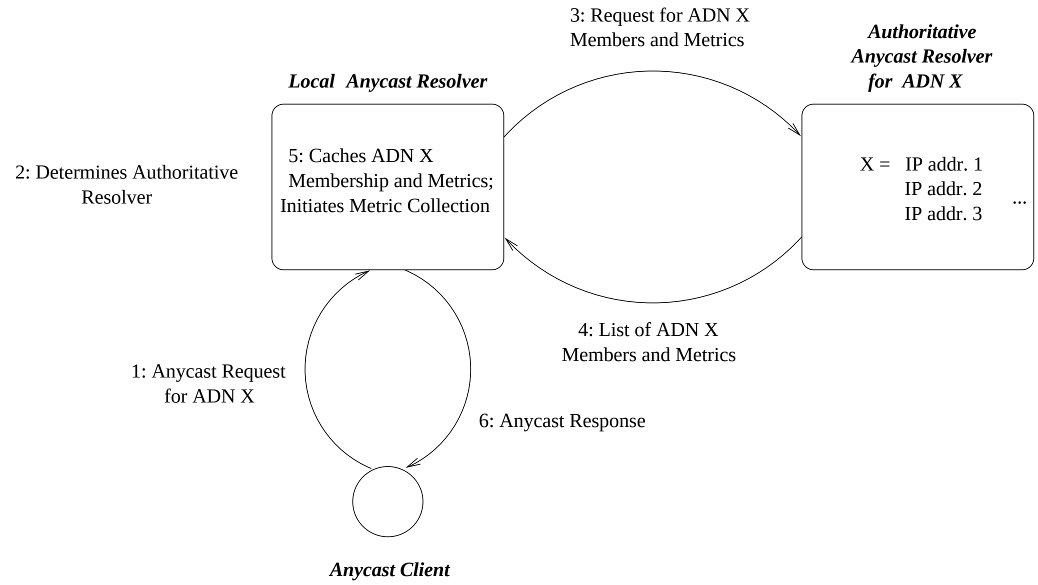
An anycast domain name (ADN) is of the form `<Service>%<Domain Name>`. Such a name will typically be used as an argument to a library call that invokes the anycasting service

[3] Although the general case of a mixture of unicast and multicast addresses is possible, we do not consider it in this paper since we cannot conceive of an application that would use it.

[4] A further generalization of our definition allows the collection of anycast IP addresses into an anycast group represented by an ADN. We relegate consideration of this "nested anycasting" to future work.

[5] Because there is no existing convention for naming multicast groups this currently only makes sense if the anycast group is a collection of hosts and not a collection of groups of hosts.

FIGURE 2 Anycast Resolver Architecture



and results in the mapping of this ADN to an IP address. The `Domain Name` part of the system indicates the location of the *authoritative* anycast resolver for this ADN. The `Service` part of the ADN identifies the service within the authoritative resolver.

The anycast resolver architecture is shown in Figure 2. Each network location is preconfigured with the address of its local anycast resolver (in the same way local DNS servers are configured). An anycast client makes its initial anycast query to its local resolver. If the resolver is authoritative for the ADN in the query or if it has cached information about the ADN, it can process the query immediately and return the appropriate repsonse. Otherwise, the local resolver determines the address of the authoritative resolver for the `Domain Name` part of the ADN and obtains the anycast group's information which is then cached in the local resolver. Determining the address of the authoritative anycast resolver for a particular domain can be done using the same technique used for DNS to determine an authoritative name server [12].

This hierarchical naming allows users to define their own anycast groups and maintain such groups in local anycast resolvers. By propagating the ADN of a locally-defined anycast group (including the name of the domain in which its authoritative resolver resides) allows others to make use of this anycast group.

An anycast resolver maintains the information necessary to perform the mapping from ADN to IP address. This information includes:

1. The list of IP addresses that form particular anycast groups. Authoritative resolvers maintain the definitive list, whereas local resolvers cache this information.

2. The metric information associated with each member of the anycast group. This information is maintained independently at each anycast resolver that has the ADN group membership information cached. Because of the local significance of the metric infor-

mation, metrics maintained at the authoritative resolver are, in general, of little value to other resolvers. The authoritative resolver may provide its locally maintained metric information whenever it receives a request from another resolver for the anycast group member list for a given ADN. Local resolvers can use this information as "hints" initially as they begin to gather their own metric information.

The information maintained in the resolver is updated using two separate mechanisms:

1. The anycast group membership information is updated according to some membership protocol. We do not discuss the details of such a protocol here because it can (and should be) dependent on the particulars of the service. For example, the list of `ftp` locations for a popular web browser can be disseminated from a home location to various anycast servers. For the web search service, on the other hand, the list of servers can be administratively configured in anycast servers.

2. The service performance metric information would typically need to be maintained dynamically. We propose and discuss some mechanisms to achieve this in Section 5.

4 Interacting with Anycast Resolvers

An anycast resolver is consulted through the use of *anycast queries* and the resolver responds with *anycast responses*. The basic anycasting query/response cycle is illustrated in Figure 1. First, the anycast client generates a query that is passed on to the anycast resolver. After processing the query, the resolver generates a response which is sent back to the client. In general, an additional processing step is performed at the client to yield the final IP address result.

Another message exchange may be required between a local resolver and an authoritative resolver. A separate (but straightforward) protocol needs to be defined for such an exchange.

4.1 Filtering and Decision Making

In our proposed approach the anycasting service is accomplished through a set of filters that are applied to the information maintained about the anycast group to obtain an IP address. We distinguish between three basic types of filters: content-independent filters, metric-based filters and policy-based filters. We further distinguish between two locations for filtering: within the resolver and at the user. (See Figure 1.)

Content-independent filters can be used to specify a selection of anycast group members based solely on membership information and not based on any other criteria maintained or known by the server. Examples of such filters include: 1) the selection of any member at random, 2) the selection of all members of the anycast group, or 3) the selection of a subset of the anycast group of some given size.

Metric-based filters specify selection according to the values of one or more metrics associated with the members of the anycast group. The following variations are of interest:

- Select the best anycast group member(s) according to a single criterion.
- Select the best anycast group member(s) according to a function of one or more metrics. For example, a weighted sum of metrics can be used, allowing a client to control the importance of each metric in the overall evaluation of an anycast group member.
- Select the best anycast group member(s) resulting from the sequential application of filters. By composing filters in series, one can specify a strict priority in the selection process. In the case of two filters, the first metric is given top priority and the second metric is used to further refine the selection set. For example, one might first select a web location that has the fastest response time, and then (amongst the fastest) choose the location that is the least number of hops away.

Policy-based filters are not based on measurements of performance, but rather encompass the broad range of other criteria that might affect the selection of an address. Policy filters are likely to be boolean, in the sense that an address either meets or fails the policy criteria. Policy filters tend to rule out groups of addresses; for example, one might be interested in accessing all servers containing income tax information that are *not* run by the IRS. Perhaps the most natural interaction of metric and policy-based filters is to first apply metrics and then apply policy, however, some other composition of the two types of filters may also be useful. Applying policy filters may be easier if the membership of the anycast group is known by domain names rather than just IP address. If the anycast group members are maintained by IP address, mapping these addresses into domain names (e.g., through the use of DNS Pointer Queries) may be required before the application of a policy filter.

Related to the issue of the *order* in which filters are applied is the issue of the *location* where they are applied. In our generalized version of anycasting, the anycast service may provide the anycast client with a *list* of addresses that meet the specified criteria. Thus there are two locations for filters: in the anycast resolver and at the anycast client. The resolver begins with the set of anycast group members and applies filters to produce a list. The client can take the resolver list and further apply filters to select a single member.

The provision of filters at the client allows for more local control, including increased privacy regarding the address selection process. Following the tax example above, one may not want to make known to the system the fact that IRS-run servers are to be avoided. Thus the policy filter that exempts IRS-run servers could be applied locally rather than specified to the system. Another example could be the use of a locally-preferred subset of servers in the form of a locally-defined anycast group. The membership of this locally-defined group is then used as a final filter to the results obtained from the anycasting service.

Obviously, the necessary information must exist at the location where the filter is applied. It seems likely that the filters exercised at the user will be predominantly policy-based, while the filters exercised in the system will be predominantly metric-based.

Successful Filtering We say that a sequence of filters applied to the set of anycast group members is *successful* if the number of addresses produced is acceptable to the client procedure. In all cases the return of exactly one IP address for an application to use should be an acceptable outcome. Ideally, the application procedure that invokes the anycasting service should also be capable of dealing with the case where multiple or no addresses are returned. This is because, in general, it is not always possible to know *a priori* the outcome of the application of a set of filters to an anycast group. In the case where multiple addresses are returned from the anycast service invocation, the application can arbitrarily pick one of the returned addresses. This is equivalent to applying yet another (content-independent) filter. In case the anycast service invocation returns no addresses, a fall back position needs to be programmed into the application procedure. For example, it could retry with another set of filters or a content-independent filter asking for any group member at random.

4.2 Filter Specification

We now turn our attention to the issue of how a client may specify a filter for the anycast resolver to use. Local filters are relatively easy to deal with since they are typically conceived, specified and applied in the same location. Resolver filters, on the other hand, are conceived at a client but are run by the resolver, which adds complexity to their specification.

We envision two types of filter specifications. In the first case, the client desires to use a filter that is already built into the resolver. In this case all the client has to do is refer to this filter using some agreed upon identifier. We expect that many popular filters will be available this way through well-known identifiers. In the second type of filter specification, the client provides a procedural description of the operation of the filter. This can be in the form of a function of metrics or a procedure describing some elaborate sequence of filter applications. An interesting possibility is the use of Java [33] as the programming language to specify filters, with the communication of the Java program to the anycast resolver included as part of anycast client/resolver interaction.

Another important issue is where and how filter specifications are made. We explore two alternatives. In the first, a new application layer interface (API) function call is provided, that can be used by an application to invoke the process of ADN to IP address mapping. Arguments to this call can be used to specify the filtering desired, either by identifying a built-in filter or by pointing to some local function or procedure to be communicated to the server. An example of this in the context of the Sockets API is the definition of a (new) `getanyhostbyname()` function call that can be used to invoke the anycasting service in a manner similar to how the `gethostbyname()` function call typically invokes the DNS service.

An alternative method for specifying filters is to use *Metric-Qualified* Anycast Domain Names (MQ-ADN) to convey the desired filter as part of the ADN being sent to the anycast server. A metric-qualified ADN is of the form `<Filter-Specification>.<ADN>` where `Filter-Specification` provides information about the filter to be used by the server. This is relatively straightforward if a built-in filter is being specified. For example, the name `ServerLoad.wwwsearch%cc.gatech.edu` could convey to the resolver the desire to use the ServerLoad built-in filter on the anycast group represented by the `wwwsearch%cc.gatech.edu`

ADN. Using metric-qualified ADNs is more complex when a procedural description of the filter is desired. One possibility is to allow complex Filter-Specification in the MQ-ADN, akin to the complex URLs generated by CGI programs [34] to perform database searches. It is necessary in such cases to have the MQ-ADNs generated automatically using library functions that are made available at the API.

MQ-ADNs can be enhanced to allow existing API calls to be left intact, thus existing applications need not be rewritten. Many of these applications take domain names as input; one would simply need to substitute a MQ-ADN instead, to allow the application to make use of the anycasting service. In this case, the MQ-ADN is augmented by the suffix ".any".

For example, a user wanting the most recent version of FreeBSD using `ftp` would ordinarily need to pick an `ftp` server and then issue the command:

> `ftp ftp.freebsd.org`

Using the augmented MQ-ADN, it would be possible for the user to issue the command:

> `ftp throughput.ftpfreebsd%cc.gatech.edu.any`
where `ftpfreebsd` is an anycast group consisting of servers that contain the desired FreeBSD software, with the membership of the anycast group maintained at the cc.gatech.edu domain's anycast resolver.

We next explain how our prototype implementation allows the use of unmodified FTP software in this instance.

4.3 An Implementation using Augmented Metric-Qualified ADNs

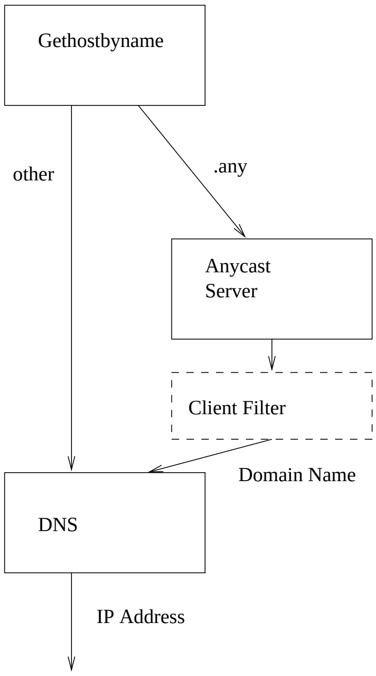
In our implementation (shown in Figure 3) we intercept calls to the `gethostbyname()` socket call and check if the argument is an ADN (i.e., ends with the `.any` suffix). If that is the case, then an anycast resolver query is formulated, after parsing the name to obtain the resolver filter specification. This query is sent to the anycast resolver which returns a set of host domain names. The set is passed through a client filter to reduce it to a single host domain name. This name is then used as input to the original `gethostbyname()` procedure to return the desired IP address. If the `gethostbyname()` argument is other than an ADN then the usual procedure is called directly. This allows us to use traditional applications without modification and gives the option of using them with or without the anycasting feature [6].

5 Metrics and Metric Determination

The use of metric-based filters in the anycasting service requires the ability to measure performance metrics with "reasonable" accuracy, without unduly loading the network or the servers.

[6] Use of this technique on existing applications requires dynamic linking of the application code to our own library. We successfully used this technique to augment many applications with anycasting capabilities, including the Mosaic web browser. However, some applications, such as the Netscape browser, are statically linked, and thus we are unable to substitute our own library subroutines.

FIGURE 3 The Structure of an Implementation using Metric-Qualified ADNs



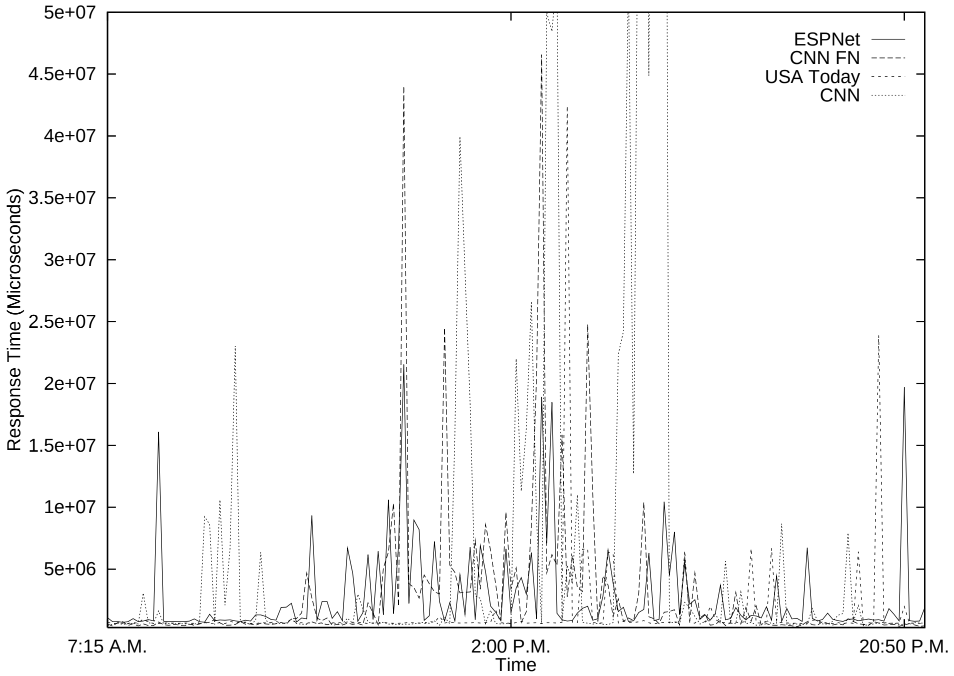
Note that the measurements need not be perfectly accurate; the anycast mechanism relies upon *relative* ordering to determine the best server, rather than absolute metric values. Further, the performance penalty associated with out-of-date or slightly inaccurate metric data will not typically be severe; rather than selecting the "best" server, the service may identify a "nearly-best" server. As a practical matter, even metrics that are sufficiently accurate to allow one to avoid the worst of the servers, e.g., those that are down or currently unreachable, will make for a useful service.

We begin this section by examining a set of metrics that may be of interest to clients in accessing replicated servers. We give examples of measurements taken on various systems; these measurements indicate that some, but not all, metrics and servers exhibit sufficient performance variation to warrant anycasting. We next turn to the issue of collecting metric information. We describe four possible ways to maintain information; these methods differ in the load that they place on the network and servers, and in the accuracy of the measurements. We examine three of the methods — proxy probing, user experience and server push — in more detail within an example environment. We conclude by summarizing the performance and system overhead characteristics of each of the methods.

5.1 Metrics

The architecture described thus far is sufficiently general to support any metric that can be measured. Metrics of interest will depend on the characteristics of the service (e.g., interactive, batch, real-time) and the preferences of the users.

FIGURE 4 HTTP Response Time at News-Oriented Servers



To better understand how anycasting will perform, we examine four specific metrics: 1) server response time, 2) server-to-user throughput, 3) server load, and 4) processor load [7]. We focus on these metrics because they are likely to be of interest to clients in selecting from a set of replicated servers. Further, they represent diversity in the components (server, path, client) involved in determining the metric value and diversity in the time scale on which the metric value changes. We envision that additional metrics (e.g., packet loss, delay jitter) would also be of interest to certain types of applications.

The use of metrics to identify the "best" server is predicated on the assumption that there are significant differences across servers at various times, otherwise the selection of a server could just as well be random. Further, the variation in a metric must occur on a time scale that is practical to track using measurement tools. We now present example measurements for each of the four metrics, with the aim of better understanding which metrics are amenable to anycasting.

5.1.1 Server Response Time

Server response time is of interest for interactive client-server applications such as web browsing. We measure server response time by measuring the round-trip time for a query [8].

[7] The original anycasting proposal [7] was primarily concerned with a server distance (hop count) metric; measuring server distance has also been studied extensively by Guyton and Schwartz [18].

[8] Specifically, a query string consisting of a set of characters not likely to be in the server's vocabulary was constructed; most often we used "Anycast Test (test version)" followed by one or more carriage returns. A

We have measured HTTP server response time at four news-oriented servers — `espnet.sportszone.com`, `www.usatoday.com`, `www.cnn.com`, and `www.cnnfn.com` — for various periods over several weeks. The measurements were taken from relatively unloaded workstations on our own research group's subnet. A typical result is depicted in Figure 4, representing server measurements every 5 minutes during a 13 hour period from 8:00 to 21:00 EST [9]. The x-axis in the plot is the probe index; the y-axis is the measured response time in microseconds. Each 25 ticks on the x-axis is approximately two hours.

These servers show time-of-day trends, with significant activity from 11:30 to 15:00 EST, or lunchtime over the three US time zones. The CNN news server is also active in the later afternoon. These results demonstrate several orders of magnitude difference between the best and worst servers at a given time, and also demonstrate that no one server is consistently best. These servers appear to be fairly good candidates for anycasting based on a response time metric.

We performed similar measurements on a set of search engines. We again observed variations in response time by an order of magnitude, ranging from about 5 seconds to 25 seconds. Interestingly, the response times at different servers were highly correlated, all tending to have roughly the same performance at a given time. We infer from this (and other data collected) that the predominant factor determining response time for these search engines is latency within the network, not load or processing at the servers. We have verified that the paths from our measurement machine to the servers share many links, thus changes in the load of these links will have a similar effect on all response times.

These experiments illustrate that some servers and metrics are not amenable to anycasting. Servers that are in a competitive business (e.g., searching) have an economic incentive to provision adequately to avoid server over-loading. A user will probably do just as well to select one of these servers arbitrarily.

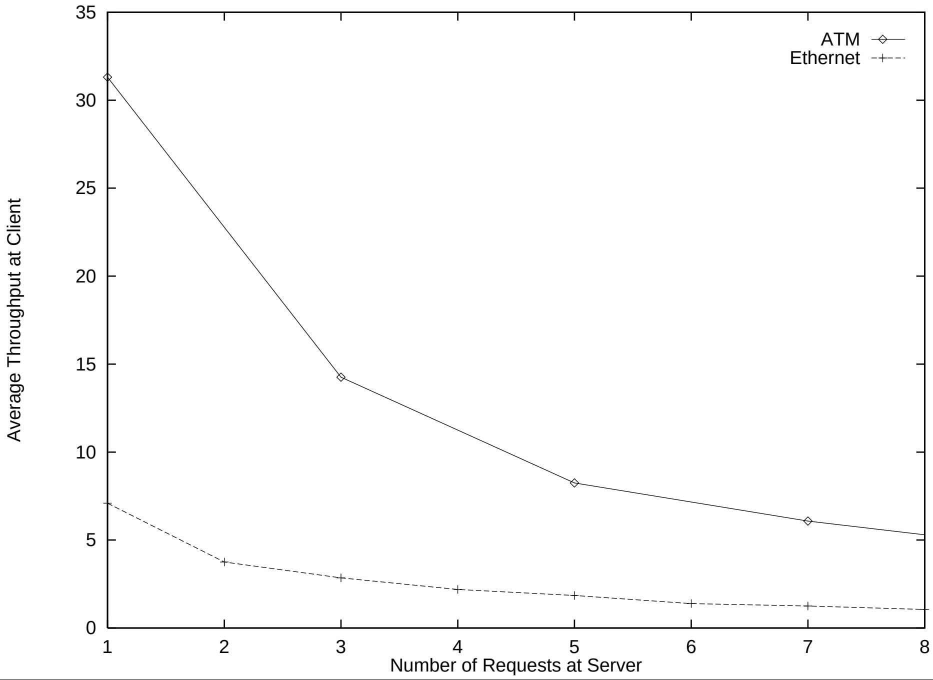
5.1.2 Server-to-User Throughput

Server-to-user throughput is a useful metric for interactive applications that must transfer significant amounts of data (e.g., web pages with graphics) and for more traditional batch-style transfers such as `ftp`. We have measured server-to-user throughput using FTP clients that

protocol specific connection was initiated to the server's specific service port (TCP port 80 in case of HTTP servers). The time to create the connection, send the query, and the time for the server to process and respond to the query were measured using the UNIX `gettimeofday()` system call. The server response was parsed to make sure the server actually responded to the query, and timeouts (if any) were logged. Thus, the query serves to measure the round trip network latency and the time for the server to fork a process and respond to the query. In the case where the server load is light, the path latency will dominate this metric. Note that a similar method could be used to measure the response time of other types of servers (e.g., FTP).

[9] The 5 minute measurement interval is accomplished by registering a clock interrupt to raise an alarm on a 5 minute interval. When the alarm occurs, interrupts are masked, the query is made and timed, then interrupts are re-enabled. It is possible to "miss" a regularly scheduled measurement if a query takes longer than the interval time. We log the timestamps of the queries so that we can detect missed measurements during post-processing of the data. These servers occasionally missed regularly scheduled probes, so the time scale is slightly off in the later part of the day.

FIGURE 5 FTP Throughput under Varying Load



we have instrumented [10]. Figure 5 demonstrates how server-to-client throughput varies as a function of server load (measured as the number of clients being served) and type of network connection between the client and the server (Ethernet or ATM). Each transfer consisted of 10 Mbytes.

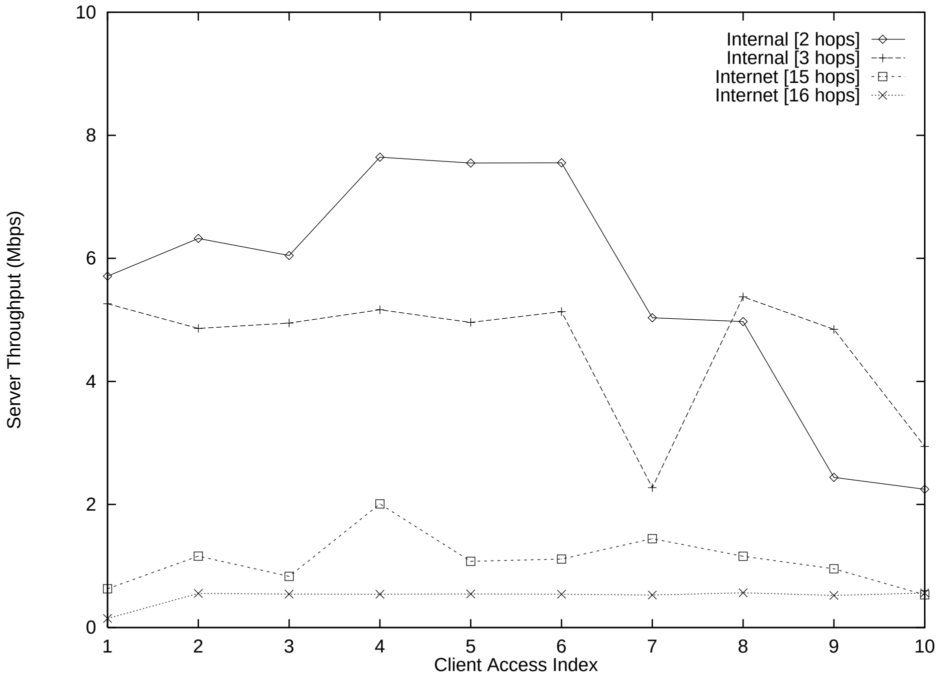
Figure 6 demonstrates variation in server-to-client throughput based on the (hop-count) distance between server and client. Ten transfers of approximately 100 Kbytes each are done for each server; the x-axis indicates the index of the transfer, and the y-axis indicates the measured throughput. These results indicate that throughput between server and client can vary considerably, both across servers and (to a lesser extent) on a single server over time. The servers that are closer to the client generally have higher throughput and greater variation in throughput over time. Note that these results suggest that hop count may make a reasonable (inverse) proxy for throughput; the further the server is from the client, the lower the throughput tends to be.

5.1.3 Server Load

Server load is a more coarse metric than response time or throughput. We include this metric because it is more practical to track, and may provide sufficient information to benefit some applications. We obtain approximate data about the server status by post-processing the logs from an http server. Specifically, the logs that we accessed contained arrival time information

[10] The client was modified so that it did not write received data to the disk; we did this to avoid having the disk write be the bottleneck in the throughput measurement.

FIGURE 6 FTP Throughput with Varying Distance



for http requests. We counted the aggregate number of arrivals in time intervals of length T seconds, and defined the server to be "loaded" if the number of requests in an interval exceeded R. Clearly the particular results will be sensitive to the choice of T and R, which should be chosen based on server capabilities and time scale of fluctuation in load.

Figure 7 shows the load status of an http server within our department (`www.cc.gatech.edu`). We have overlaid results from three different days — Saturday, May 11 - Tuesday, May 14, 1996. These results use time interval $T = 60$ seconds and threshold $R = 85$ requests. We note that the activity is distinctly different on the weekend days versus the weekdays. Further, all four traces exhibit time-of-day variations.

5.1.4 Processor Load

Measurement of processor load is a useful metric for an application that is quite different from the server access described thus far. For example, a process manager responsible for distributing tasks onto machines can view a set of processors as a replicated service, with selection of the best machine corresponding to the least loaded processor.

We have experimented with measuring processor load for various machines on our campus network. Figure 8 shows typical result depicting the load (as reported by `rup`) on two compute servers (`forge` and `lennon`) over a typical 24 hour period, with measurements taken every 5 minutes. This portion of the data runs from about 2am to 2am. These values also show time-of-day trends, with one of the servers (`forge`) seeing significant activity in the middle of the night (roughly 2am). To properly compare the loads on the various machines, the data

FIGURE 7 Load Status of HTTP Server

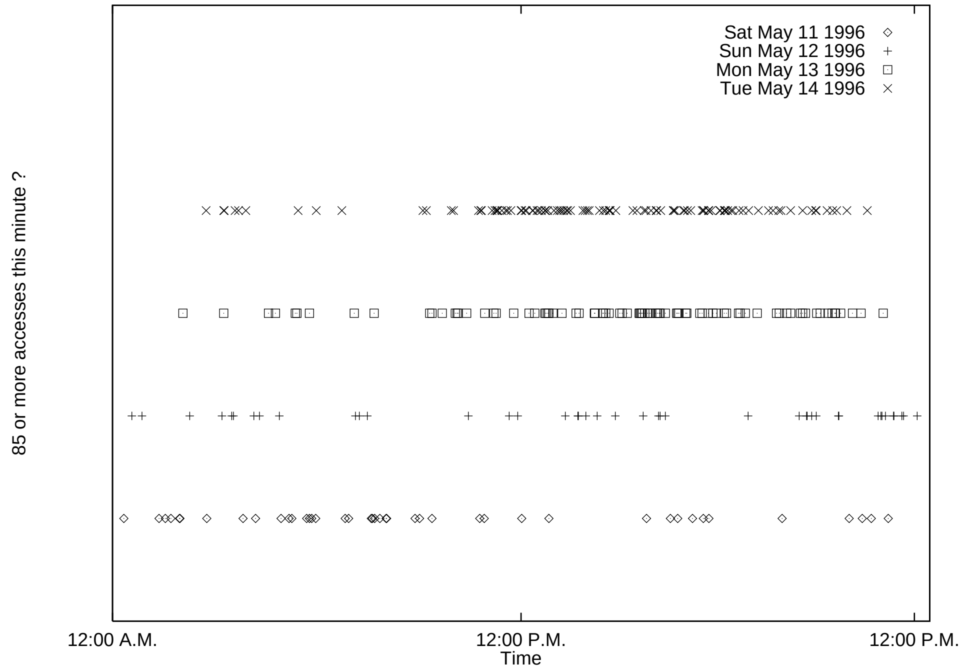
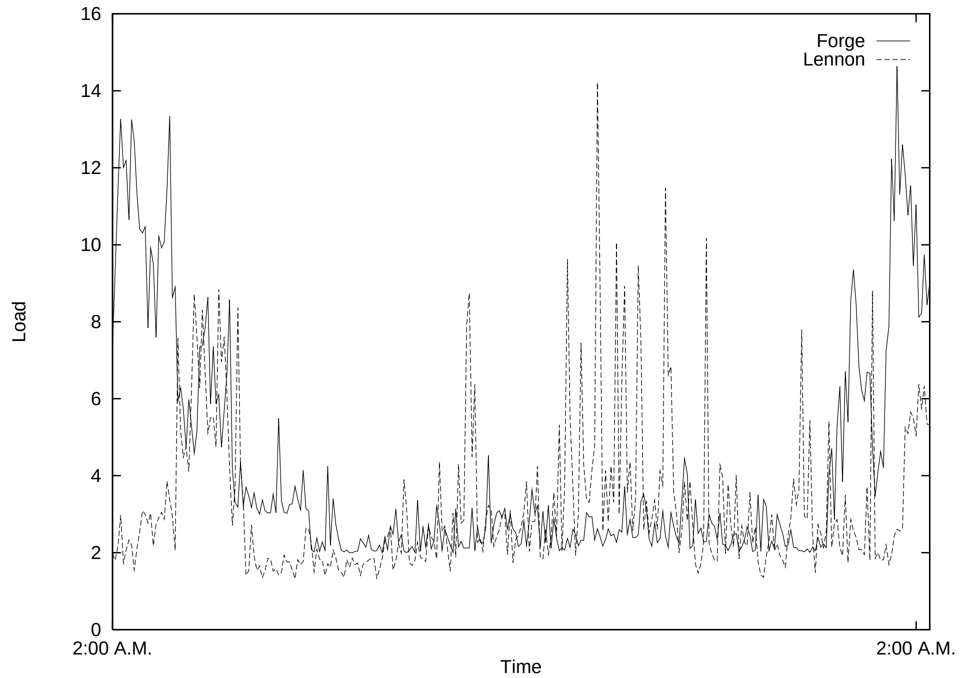


FIGURE 8 Processor Load on Compute Servers



should be normalized using some measure of processing capability, such as number and type of processors. In this example both machines have four processors each.

5.2 Metric Collection Techniques

The previous subsection indicates that there are servers and metrics that can benefit from anycasting. We next consider how to efficiently and accurately maintain databases of the metric values. We have identified four possible approaches to maintaining replicated server performance information in anycast servers' database:

1. *Remote Server Performance Probing:* In this technique probing agents are in charge of periodically querying the replicated servers to determine the performance that will be experienced if a client were to request service. Each probing agent acts as a proxy for a set of clients. These probes need to be designed to mimic (as much as possible) the parts of the network and the server that the client request will "exercise". We have experimented with this technique in our implementation. In Section 5.3 we give an example of our results and examine the issue of probe location and accuracy.

2. *Server Push:* It may be advantageous in some circumstances to have the replicated servers send (or push) the relevant performance information (as measured locally) onto the anycast servers. The primary advantage of this technique is scalability: it allows the server to update its performance information only when interesting changes are observed. Further, the update information can be (network layer) multicast to all anycast resolvers that maintain information about the server. The anycast resolvers can join well-known multicast groups for each server that they are interested in, allowing the servers to disseminate performance information without knowing the identities of the resolvers. The multicast groups may be organized in other ways, for example with one multicast group per anycast domain name, or one multicast group per anycast group. These alternatives differ in the number of multicast groups required, the ease of maintaining the multicast groups and the amount of network traffic required for disseminating server information.

   We have considered several variations on server push based on the type and/or frequency of information that is pushed. In Section 5.4 we describe an experiment that considers pushing good news, pushing bad news and periodically pushing current state.

3. *Probing for Locally-Maintained Server Performance:* Another possibility is to have each replicated server maintain its own locally monitored performance metrics in a globally readable file. Remote probing locations can then read the information in the file (as opposed to attempting to exercise the server) to obtain the desired information. In a sense, this is a hybrid between the probing-for-performance and the server-push techniques. Since probes merely read from a locally-maintained file, they may represent less of a burden on the server than the probing-for-performance approach.

4. *User Experience:* The last technique is motivated by the observation that users currently make server access decisions based, in part, on past experience. That is, if one finds a particular server to be unreachable, that server is likely to be avoided for a period of time.

> Collecting information about past experience offers a coarse method of maintaining server performance. The primary advantage of this method is that the information is collected for free; no additional burden is placed on the server or the network. The quantity and accuracy of the information can be increased by sharing of experience among clients. For example, a gateway into a campus might maintain server performance information based on the experience of all clients on the campus.

We next describe several experiments that examine performance and system overhead costs associated with three of these techniques.

5.3 Example: Proxy Probing and User Experience

Ideally, the probes which collect metric information would be run in as few locations as possible yet still provide accurate data to be used in filtering. Each probe puts load onto the network and the server, thus fewer probes will cause less overhead for metric monitoring. Again, it is not necessary that the absolute values of the metrics be precise, only that the relative comparison reflect the view at the user.

Metrics which are dependent on the path from the user to the server will need to be probed from a location that includes (some part of) the path. At the extreme, these probes could be run at each user, and thus would follow a path from the user to the server. For scalability, it is preferable to run path-dependent probes on "proxies" that provide metric data for a group of users. These proxies might be chosen to cover all hosts on a subnetwork, within a routing domain or within a geographic region. As a proxy covers a larger set of hosts, the number of proxies (and thus network load due to probing) decreases, however so does the accuracy of the proxy data relative to an arbitrary user in the group.

The experience of a client (or a group of clients that share information) can be used to maintain metric data with no additional load on the network or servers. We expect metric data that is maintained using experience to be less accurate than data that is maintained by a more regular measurement process. There are two sources for the inaccuracy: (1) the servers may not be accessed regularly, and (2) the experience at one client may not accurately reflect performance at another client. We note that this second form of inaccuracy is essentially the same as occurs in the proxy probing method. Thus the experiment described next also provides some information about the accuracy of user experience.

We have explored the question of proxy location and accuracy of the proxy data using the following experimental setup. We ran probes on five machines, three on our local campus network and two on the network of another campus [11]. We measured HTTP server response time to a set of seven servers located around the world. Five of them are FreeBSD servers; the other two were chosen to get good geographical distribution [12]. Measurements were taken

[11] The two campuses are separated by several states and 3-4 network hops. These experiments should be run with a more diverse set of machine locations, however we were limited in our access to widely dispersed systems.

[12] The servers we probed were:
minnie.cs.adfa.oz.au, www.br.freebsd.org, www.ca.freebsd.org, www.de.freebsd.org, www.freebsd.org, www.jp.freebsd.org, www.kr.freebsd.org, cc2.iitb.ernet.in, and www.wustl.edu.

	Best	Second	Third	Fourth	Fifth	Sixth	Seventh
`local1`	100.00	0.00	0.00	0.00	0.00	0.00	0.00
`local2`	42.00	40.00	8.50	2.00	6.00	1.00	0.50
`local3`	34.00	36.50	18.00	7.00	4.00	0.50	0.00
`remote1`	26.50	29.50	20.50	12.00	7.50	2.50	1.50
`remote2`	20.00	30.00	34.00	10.50	4.00	0.50	1.00

Table 1: Accuracy of Proxies

every three minutes for about 13 hours from 23:00 EST to 12:00 EST on the following day.

In order to compare the accuracy of different proxies, the times at which the readings are taken must be relatively synchronized. (That is, we would like to isolate the issue of inaccuracy caused by taking the measurements at different *locations* from the inaccuracy caused by taking the measurements at different *times*.) This is accomplished by starting the processes as quickly as possible from a machine running shells on every proxy. More elaborate synchronization is certainly possible, however an inspection of the time logs indicates that the probes are occurring within seconds of one another (modulo differences in system clocks between the two campuses where the proxies were located).

In this set of data, we did find that the proxies occasionally missed regularly scheduled probes. We post-processed the data files to find the misses at each proxy and remove the corresponding time values from the data files for the other proxies. The "clean" versions of the data files contained time-consistent probe measurements across all proxies.

To assess the accuracy of the proxies, we designate one machine as the "base" and all others as proxies. For each measurement point, we determine the best server as selected by the base and by each proxy, based on the measurement data. We then record how often the proxies agree or disagree with the base on the selection of the best server. More precisely, we count how often each proxy agrees with the base (i.e., selects the same best server), how often the proxy selects the second best base server, and so on. Table 1 shows the percentage of time that each proxy picks each server.

The machines labeled `local1`, `local2` and `local3` are three hosts on different subnets within our own local campus network. `local1` is the base machine. `remote 1` and `remote2` are two machines on different subnets of the remote campus' network. The proxies on the local campus agree more often and more accurately with the base than the proxies on the remote campus. Even the proxies on the remote campus give fairly good results, selecting the best or second best server about 50% of the time. (Note that a random selection of servers — as might be obtained without anycasting support — would pick the best server 1/7th or 14% of the time.)

The results are affected by whether the servers are relatively near the measurement machines or far away. We have divided the analysis above into two parts: one containing the three North American servers and the other containing the four foreign servers. Tables 2 and 3 show the results for these two groups of servers. The advantage of nearby proxies is more significant for

	Best	Second	Third
`local1`	100.00	0.00	0.00
`local2`	67.50	20.50	12.00
`local3`	45.5	40.50	14.00
`remote1`	41.50	26.00	32.50
`remote2`	37.50	25.00	37.50

Table 2: Accuracy of Proxies - North American Servers

	Best	Second	Third	Fourth
`local1`	100.00	0.00	0.00	0.00
`local2`	62.50	27.50	6.00	4.00
`local3`	65.50	26.50	5.00	3.00
`remote1`	52.50	41.00	4.00	2.50
`remote2`	57.00	33.50	7.00	2.50

Table 3: Accuracy of Proxies - Foreign Servers

the North American servers; the paths from the proxies to these servers are not as long as to the foreign servers, and have more commonality for the nearby proxies.

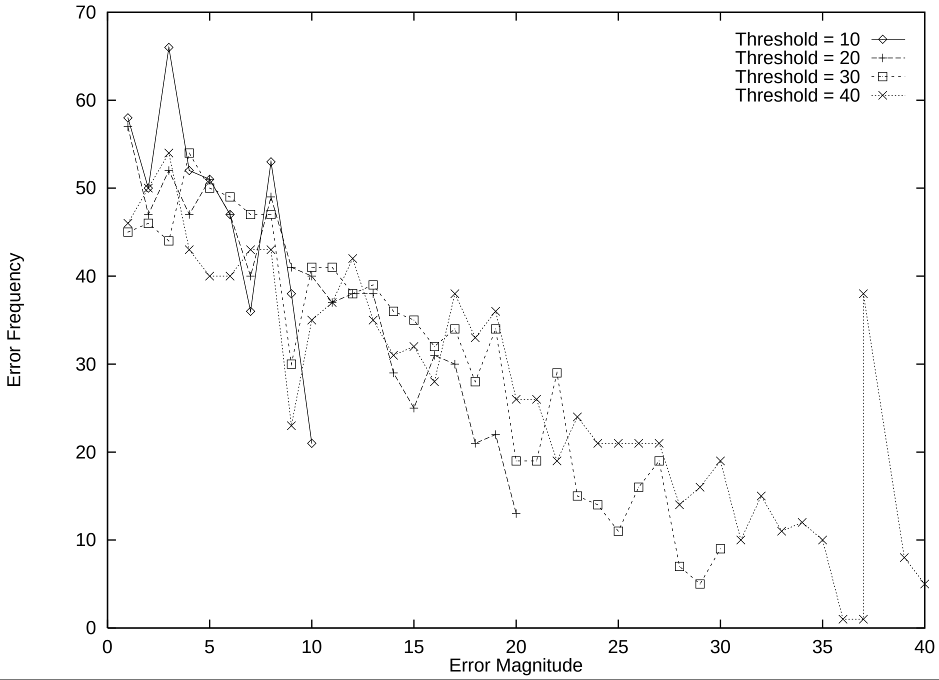
All of the proxies do quite well for the foreign servers, with each proxy agreeing with the base over 50% of the time. The routes to the foreign machines have many hops in common; for example, all machines share the same last 15 hops to the server in India.

5.4 Example: Server Push

To discuss the performance of the server push technique, we must first define the algorithm that the server will use to determine what information to push and when. In general, we want the server to push state information whenever the state has changed sufficiently to be "interesting" with some constraint on the maximum frequency of updates so as to bound the overhead of the updating mechanism.

Note that the task of updating link state in a distributed routing environment has precisely the same criteria. We have adopted the link state update algorithm used in the ARPANET [35] and experimented with the performance and overhead with a variety of parameters. The update algorithm is parameterized by a measurement interval I, a maximum threshold T and a reduction factor R. The algorithm maintains a current threshold C, initialized to T. The server measures its state over each interval I. If the state changes from the previous measurement by at least C, the state is pushed and C is reset to T. If the state does not change by at least C, C is reduced by R. (Note that when C becomes 0, the state will be pushed and C will be reset to T.) The algorithm will send updates at least every T/R time units and at most every I time

FIGURE 9 Performance of Server Push Algorithms



units.

We have implemented this server push algorithm for an HTTP server, where the measurement quantity is the number of connections initiated in the measurement interval. In our experiments we use $I = 1$ minute. We process HTTP server logs to determine the pushed values for varying T and R.

In Figure 9 we examine the accuracy of the updating mechanism as a function of T and R. At each one minute interval, we compare the value of the server load based on the pushed information to the true value of load extracted from the logs. We record the magnitude of the difference between the estimate and the true value, and plot a frequency distribution. In Figure 9 we used $R = 5$ minutes and varying T; we also collected data for $R = 10$ minutes.

Table 4 gives the average value of the inaccuracy and the number of update messages, for various values of T and R. The number of update messages generated is a measure of the load placed on the network by the mechanism; the average value of the inaccuracy summarizes the performance data given by the frequency distribution. The tradeoff is clear: a more accurate algorithm also incurs a larger overhead. Interestingly, increasing the maximum interval between updates from 5 to 10 has relatively little effect on the accuracy for a constant update threshold.

5.5 Comparison of Metric Collection Techniques

Which technique is most appropriate will depend on a number of factors, including the time-scale on which the metric varies, the components (server, network path, client) that determine

Update Threshold	Max. Interval Between Updates	Average Error	Number of Updates
10	5	1.53	939
10	10	1.64	922
20	5	4.21	682
20	10	4.69	643
30	5	6.87	537
30	10	7.74	464
40	5	9.21	445
40	10	10.45	360
50	5	10.82	291
50	10	12.68	379

Table 4: Load and Accuracy of Server Push

	Net Load	Server Mod	Server Load	Exercises Net Path	Accuracy
Probing	$2PT_p$	No	Moderate	Yes	Moderate
Server Push	T_s	Yes	Low	No*	High
Reading Server Log	PT_p	Yes	Low	No*	High
User Experience	None	No	None	Yes	Low/Varies

(* See note in text)

Table 5: Comparison of Metric Collection Techniques

the metric value, the desired accuracy of the information, and the cost of burdening the network and/or the server.

Table 5.5 summarizes the four techniques based on performance and cost dimensions. The first three columns are measures of system overhead. The Net Load column represents the number of messages generated per unit time to obtain the metric data from one server, where P is the number of proxies, T_p is the period of proxy probing, T_s is the period of server push. Note that the Server Push messages are multicast rather than unicast. The Server Mod column indicates whether the server must be modified to allow the metric to be collected. The Server Load column expresses (relatively) how much additional load is placed on the server by the collection of the metric data. The last two columns are performance measures, indicating whether the method exercises network path, and (relatively) how accurately the method is able to maintain the metrics that it can evaluate. Note that the Server Push and Server Log methods can be made to exercise the network path by measuring hop count or latency for the metric update message.

6 Concluding Remarks

We have explored the implications of an anycasting service supported at the application layer. Specifically, we have developed and evaluated an implementation based on the use of anycast servers to map anycast domain names to one or more IP addresses. Via metric and policy filters, we explicitly provide for considerable user control over the selection of a specific server from a group, while preserving the spirit of the anycast definition. That is, clients control the selection by specifying desired properties of the server (e.g., fastest response time), and need not know which specific servers are able to satisfy the request.

We believe that one advantage of our application layer support is its ability to operate with current network layer implementations. Further, our work is complemented by other efforts in resource discovery and efficient indexing of information.

In the area of metric determination, our results suggest a number of avenues for future work. Collecting proxy and user experience data from a larger set of sites would allow us to better determine how accuracy varies with proxy, server, and base machine locations. It would also be interesting to explore methods for applying base-machine-specific transformations to the proxy or user-experience data to improve accuracy. We have examined a simple class of server push algorithms; more work is needed to optimize these algorithms, and may require metric-specific techniques. Our results on throughput indicate that there may be metrics (e.g., hop count) that are easier to measure accurately and still correlate well with end-user perceived performance.

References

[1] P. B. Danzig, D. Delucia, and K. Obraczka, "Massively replicating services in wide-area internetworks," tech. rep., University of Southern California, 1994.

[2] P. Danzig, R. Hall, and M. Schwartz, "A case for caching file objects inside internetworks," in *Proceedings of SIGCOMM 93*, pp. 239–248, 1993.

[3] J. Pitkow and M. Recker, "A simple yet robust caching algorithm based on dynamic access patterns," in *Proceedings of 2nd WWW conference*, 1994.

[4] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of ACM Multimedia 94*, pp. 15–23, 1994.

[5] D. Gifford, "Polychannel systems for mass digital communication," *Communications of the ACM*, vol. 33, pp. 1847–1851, February 1990.

[6] R. Clark and M. Ammar, "Providing scalable web service using multicast delivery," in *Proceedings of 2nd IEEE Workshop on Services in Distributed and Networked Environments*, pp. 19–26, 1995.

[7] C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," *RFC 1546*, November 1993.

[8] K. Birman and T. Joseph, "Reliable communication in the presence of failures," *ACM Transactions on Computer Systems*, vol. 5, pp. 47–76, February 1987.

[9] R. Hinden and S. Deering, "IP version 6 addressing architecture," *RFC 1884*, December 1995.

[10] J. Bernabeu, M. Ammar, and M. Ahamad, "Optimizing a generalized polling protocol for resource finding over a multiple access channel," *Computer Networks and ISDN Systems*, vol. 27, pp. 1429–1445, 1995.

[11] D. Oppen and Y. Dalal, "The clearinghouse: A decentralized agent for locating named objects in a distributed environment," *ACM Transactions on Office Information Systems*, vol. 3, pp. 230–253, July 1983.

[12] P. Mockapetris, "Domain names – concepts and facilities," *RFC 1034*, November 1987.

[13] I. Gopal and A. Segall, "Directories for networks with casually connected users," in *Proceedings of INFOCOM 88*, pp. 1060–1064, 1988.

[14] A. Birrel, R. Levin, and M. Schroeder, "Grapevine: An exercise in distributed computing," *Communications of the ACM*, vol. 25, pp. 260–274, April 1982.

[15] D. Terry, "Caching hints in distributed systems," *IEEE Transactions on Software Engineering*, vol. 13, pp. 48–54, January 1987.

[16] R. Fowler, *Decentralized Object Finding Using Forwarding Addresses.* PhD thesis, University of Washington, 1985.

[17] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service location protocol," *Internet Draft (work in progress) draft-ietf-srvloc-protocol-13.txt*, June 1996.

[18] J. Guyton and M. Schwartz, "Locating nearby copies of replicated Internet servers," in *Proceedings of SIGCOMM 95*, pp. 288–298, 1995.

[19] C. M. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz, and D. Wessels, "Harvest: A scalable, customizable discovery and access system," Tech. Rep. CU-CS-732-94, University of Colorado - Boulder, 1995.

[20] K. Moore, J. Cox, and S. Green, "SONAR - a network proximity service," *Internet Draft (work in progress) draft-moore-sonar-01.txt*, February 1996.

[21] V. Jacobson, "Traceroute." available from ftp://ftp.ee.lbl.gov/traceroute.tar.Z.

[22] A. Ghosh and P. Brooks, "Mwatch 3.6.2." available from http://www.cl.cam.ac.uk/mbone/index.html#Mrouted.

[23] D. Wood, S. Coleman, and M. Schwartz, "Fremont: A system for discovering network characteristics and problems," in *Proceedings of 1993 Winter USENIX Conference*, pp. 335–347, January 1993.

[24] B. Schroeder, "On-line monitoring: A tutorial," *IEEE Computer*, vol. 28, pp. 72–78, June 1995.

[25] F. Lange, R. Kroeger, and M. Gergeleit, "Jewel: Design and implementation of a distributed measurement system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 657–671, Novemeber 1992.

[26] J. Gwertzman and M. Seltzer, "The case for geographical push caching," Tech. Rep. 34-94, Harvard University, 1994.

[27] M. Humes, "Netscape's server push, client pull and CGI animation." http://www.emf.net/mal/animate.html.

[28] E. D. Katz, M. Butler, and R. McGrath, "A scalable HTTP server: The NCSA prototype," *Computer Networks and ISDN Systems*, vol. 27, pp. 155–164, 1994.

[29] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Networks*, vol. 2, pp. 95–114, 1978.

[30] D. Gifford, "Weighted voting for replicated data," in *Proceedings of 7th Symposium on Operating Systems*, pp. 150–162, ACM, 1979.

[31] M. Ahamad and M. H. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *IEEE Transactions on Software Engineering*, vol. 15, pp. 492–496, April 1989.

[32] D. Barbara and H. Garcia–Molina, "Mutual exclusion in partitioned distributed systems," *Distributed Computing*, vol. 1, pp. 119–132, 1986.

[33] J. Gosling and H. McGibon, "The Java language environment: A white paper." http://www.javasoft.com/whitePaper/java-whitepaper-1.html.

[34] "The common gateway interface." http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[35] E. C. Rosen, "The updating protocol of arpanet's new routing algorithm," *Computer Networks*, no. 4, pp. 11–19, 1980.