Application-Layer Anycasting

Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura
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

Abstract

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. An expanded version of this work can be found as a technical report.[1].

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 networked services is the ability to meet the demands of a large number of geographically wide-spread users. It is also important that the user-perceived quality of service (e.g., response time, throughput, reliability) be maintained at an acceptable and (in the case of commercial services) competitive 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]. 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 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 unicast, broadcast, multicast communication paradigms. In particular, we differentiate between the anycasting *service definition* and the *protocol layer* providing the anycasting service [2]. 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 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

[1] ftp://ftp.cc.gatech.edu/pub/coc/tech_reports/1996/GIT-CC-96-25.ps.Z

[2] 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. 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.

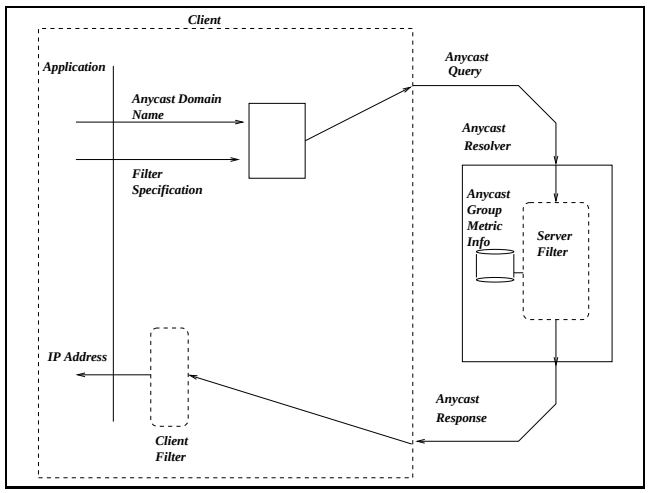

FIGURE 1: Anycast Name Resolution Query/Response Cycle

and forward packets 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 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. Protocols that require all datagrams be delivered to a single host may use anycasting for discovery of a satisfactory host but use standard IP for subsequent transmissions.
- The network layer is able to efficiently determine shortest paths, thus it is well-suited for an anycasting service that selects the closest server based upon a shortest path metric such as hop count. An application layer approach is better suited at handling a variety of other metrics such as server throughput.

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 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* containing performance data about servers. The performance data is used in the selection of a server from a group, based upon user-specified performance criteria.

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. Techniques were proposed and investigated, including: 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]), 3) the use of caching of a resource's location (not content) at sites where the resource is frequently accessed [15]. 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].

The Internet has experienced a dramatic growth in the use and provision of information services such as `ftp`, `archie`, `gopher`, and more recently the World Wide Web. This has resulted in heavy demands being placed on servers and 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.

To avoid the scalability problems inherent in prob-

ing for performance, we 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 [20] where servers are in charge of pushing the desired information onto remote caches and the *server push* mechanism [21] implemented in Netscape browsers.

Finally, anycasting is related to the technique used to build scalable HTTP servers [22]. In such a scheme multiple servers are clustered and appear to the outside world as a single logical server. Modifications to the DNS resolution mechanisms are made to distribute the load among the servers.

3 Generalized Anycasting

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

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 news sites (e.g., CNN Interactive, Time Magazine and USA Today) carry different information, they, nevertheless, can be construed as being replicated servers for some applications.

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. In selecting the best server, a performance measure of interest might be CPU load. An interesting variation occurs if the user has a parallel program to run that requires the use of multiple hosts. This is an 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. 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 are 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 classes of quorum consensus applications [23, 24, 25, 26]. 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 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 [4]. 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

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

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

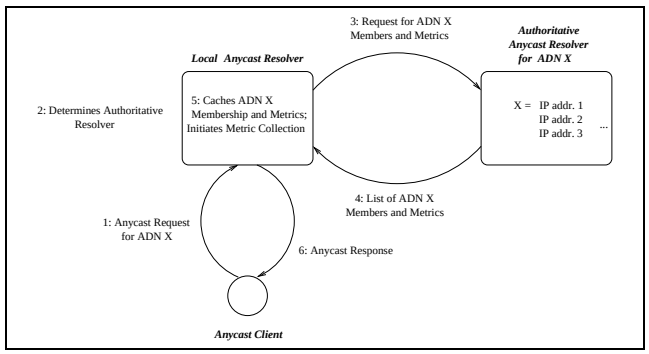

FIGURE 2: Anycast Resolver Architecture

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.

We propose that an anycast domain name (ADN) be 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 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 response. 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. 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 information, 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.

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. 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. 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.

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

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 specified 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 a sequence of filter applications.

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 informa-

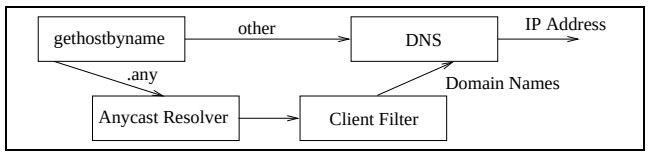

FIGURE 3: Implementation using Metric-Qualified ADNs

tion 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.wwwnews%cc.gatech.edu` could convey to the resolver the desire to use the ServerLoad built-in filter on the anycast group represented by the `wwwnews%cc.gatech.edu` ADN.

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".

4.3 Implementation Using MQ-ADNs

In our implementation (shown in Figure 3) we intercept calls to the `gethostbyname()` library[5] 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. 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.

5.1 Metrics

To better understand how anycasting will perform, we have used 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.

Detailed experiments, results, and perfomance analysis are presented in the technical report [27].

5.2 Metric Collection Techniques

We now 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 actually 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 [27].

2. *Server Push:* It may be advantageous in some circumstances to have the replicated servers send (or push) the relevant local performance information onto the anycast servers. The primary advantage

[5] in SunOS 4.1.3

[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.

[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].

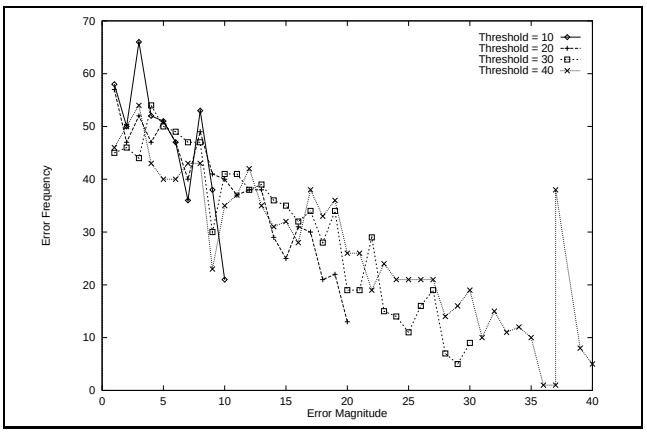

FIGURE 4: Performance of Server Push Algorithms

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 1: Load and Accuracy of Server Push

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.

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.

5.3 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 [28] 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 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

	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	$2PT_p$	Yes	Low	No*	High
User Experience	None	No	None	Yes	Low/Varies

(* See note in text)

Table 2: Comparison of Metric Collection Techniques

$I = 1$ minute. We process HTTP server logs to determine the pushed values for varying T and R.

In Figure 4 we examine the accuracy of the updating mechanism as a function of T. 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 of error magnitudes for each threshold value. Thus, the plots represent the error distribution for different values of T, and each data point is the number of times the estimate had an error of the corresponding error magnitude. In Figure 4 we used $R = 5$ minutes and varying T; we also provide results for $R = 10$ minutes.

Table 1 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.

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

Table 2 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.

7 Acknowledgements

We would like to thank Yusuf Goolamabbas for his involvement in the original implementation of Application-Layer Anycasting.

References

[1] P. B. Danzig, D. Delucia, and K. Obraczka, "Massively replicating services in wide-area internetworks." Tech. Rep., University of Southern California, January 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] J. Gwertzman and M. Seltzer, "The case for geographical push caching," Tech. Rep. 34-94, Harvard University, 1994.

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

[22] 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.

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

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

[25] 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.

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

[27] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei, "Application level anycasting," Tech. Rep. 96-25, College of Computing, Georgia Institute of Technology.

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