OASIS: Anycast for Any Service

Michael J. Freedman[*‡], Karthik Lakshminarayanan[†], David Mazières[‡]
[*]New York University, [†]U.C. Berkeley, [‡]Stanford University
http://www.coralcdn.org/oasis/

Abstract

Global anycast, an important building block for many distributed services, faces several challenging requirements. First, anycast response must be fast and accurate. Second, the anycast system must minimize probing to reduce the risk of abuse complaints. Third, the system must scale to many services and provide high availability. Finally, and most importantly, such a system must integrate seamlessly with unmodified client applications. In short, when a new client makes an anycast query for a service, the anycast system must ideally return an accurate reply without performing any probing at all.

This paper presents OASIS, a distributed anycast system that addresses these challenges. Since OASIS is shared across many application services, it amortizes deployment and network measurement costs; yet to facilitate sharing, OASIS has to maintain network locality information in an application-independent way. OASIS achieves these goals by mapping different portions of the Internet in advance (based on IP prefixes) to the geographic coordinates of the nearest known landmark. Measurements from a preliminary deployment show that OASIS, surprisingly, provides a significant improvement in the performance that clients experience over state-of-the-art on-demand probing and coordinate systems, while incurring much less network overhead.

1 Introduction

Many Internet services are distributed across a collection of servers that handle client requests. For example, high-volume web sites are typically replicated at multiple locations for performance and availability. Content distribution networks amplify a website's capacity by serving clients through a large network of web proxies. File-sharing and VoIP systems use rendezvous servers to bridge hosts behind NATs.

The performance and cost of such systems depend highly on the servers that clients select. For example, file download times can vary greatly based on the locality and load of the chosen replica. Furthermore, a service provider's costs may depend on the load spikes that the server-selection mechanism produces, as many data centers charge customers based on the 95th-percentile usage over all five-minute periods in a month.

Unfortunately, common techniques for replica selection produce sub-optimal results. Asking human users to select the best replica is both inconvenient and inaccurate. Round-robin and other primitive DNS techniques spread load, but do little for network locality.

More recently, sophisticated techniques for server-selection have been developed. When a legacy client initiates an anycast request, these techniques typically probe the client from a number of vantage points, and then use this information to find the closest server. While efforts, such as virtual coordinate systems [6, 28] and on-demand probing overlays [40, 46], seek to reduce the probing overhead, the savings in overhead comes at the cost of accuracy of the system.

Nevertheless, significant on-demand probing is still necessary for all these techniques, and this overhead is reincurred by every new deployed service. While on-demand probing potentially offers greater accuracy, it has several drawbacks that we have experienced first-hand in a previously deployed system [10]. First, probing adds latency, which can be significant for small web requests. Second, performing several probes to a client often triggers intrusion-detection alerts, resulting in abuse complaints. This mundane problem can pose real operational challenges for a deployed system.

This paper presents OASIS (*O*verlay-based *A*nycast *S*ervice *I*nfra*S*tructure), a shared locality-aware server selection infrastructure. OASIS is organized as an infrastructure overlay, providing high availability and scalability. OASIS allows a service to register a list of servers, then answers the query, “Which server should the client contact?” Selection is primarily optimized for network locality, but also incorporates liveness and load. OASIS can, for instance, be used by CGI scripts to redirect clients to an appropriate web mirror. It can locate servers for IP anycast proxies [2], or it can select distributed SMTP servers in large email services [26].

To eliminate on-demand probing when clients make anycast requests, OASIS probes clients in the background. One of OASIS's main contributions is a set of

Keyword	Threads	Msgs	Keyword	Threads	Msgs
abuse	198	888	ICMP	64	308
attack	98	462	IDS	60	222
blacklist	32	158	intrusion	14	104
block	168	898	scan	118	474
complaint	216	984	trojan	10	56
flood	4	30	virus	24	82

Figure 1: Frequency count of keywords in PlanetLab *support-community* archives from 14-Dec-04 through 30-Sep-05, comprising 4682 messages and 1820 threads. Values report number of messages and unique threads containing keyword.

techniques that makes it practical to measure the entire Internet in advance. By leveraging the locality of the IP prefixes [12], OASIS probes only each prefix, not each client; in practice, IP prefixes from BGP dumps are used as a starting point. OASIS delegates measurements to the service replicas themselves, thus amortizing costs (approximately 2–10 GB/week) across multiple services, resulting in an acceptable per-node cost.

To share OASIS across services and to make background probing feasible, OASIS requires *stable network coordinates* for maintaining locality information. Unfortunately, virtual coordinates tend to drift over time. Thus, since OASIS seeks to probe an IP prefix as infrequently as once a week, virtual coordinates would not provide sufficient accuracy. Instead, OASIS stores the geographic coordinates of the replica closest to each prefix it maps.

OASIS is publicly deployed on PlanetLab [34] and has already been adopted by a number of services, including ChunkCast [5], CoralCDN [10], Na Kika [14], OCALA [19], and OpenDHT [37]. Currently, we have implemented a DNS redirector that performs server selection upon hostname lookups, thus supporting a wide range of unmodified client applications. We also provide an HTTP and RPC interface to expose its anycast and locality-estimation functions to OASIS-aware hosts.

Experiments from our deployment have shown rather surprisingly that the accuracy of OASIS is competitive with Meridian [46], currently the best on-demand probing system. In fact, OASIS performs better than all replica-selection schemes we evaluated across a variety of metrics, including resolution and end-to-end download times for simulated web sessions, while incurring much less network overhead.

2 Design

An anycast infrastructure like OASIS faces three main challenges. First, network peculiarities are fundamental to Internet-scale distributed systems. Large latency fluctuations, non-transitive routing [11], and middleboxes such as transparent web proxies, NATs, and firewalls can produce wildly inaccurate network measurements and hence suboptimal anycast results.

Second, the system must balance the goals of accuracy, response time, scalability, and availability. In general, using more measurements from a wider range of vantage points should result in greater accuracy. However, probing clients on-demand increases latency and may overemphasize transient network conditions. A better approach is to probe networks in advance. However, services do not know which clients to probe apriori, so this approach effectively requires measuring the whole Internet, a seemingly daunting task.

A shared infrastructure, however, can spread measurement costs over many hosts and gain more network vantage points. Of course, these hosts may not be reliable. While structured peer-to-peer systems [39, 42] can, theoretically, deal well with unreliable hosts, such protocols add significant complexity and latency to a system and break compatibility with existing clients. For example, DNS resolvers and web browsers deal poorly with unavailable hosts since hosts cache stale addresses longer than appropriate.

Third, even with a large pool of hosts over which to amortize measurement costs, it is important to minimize the rate at which any network is probed. Past experience [10] has shown us that repeatedly sending unusual packets to a given destination often triggers intrusion detection systems and results in abuse complaints. For example, PlanetLab's *support-community* mailing list receives thousands of complaints yearly due to systems that perform active probing; Figure 1 lists the number and types of complaints received over one ten-month period. They range from benign inquiries to blustery threats to drastic measures such as blacklisting IP addresses and entire netblocks. Such measures are not just an annoyance; they impair the system's ability to function.

This section describes how OASIS's design tackles the above challenges. A two-tier architecture (§2.1) combines a reliable core of hosts that implement anycast with a larger number of replicas belonging to different services that also assist in network measurement. OASIS minimizes probing and reduces susceptibility to network peculiarities by exploiting *geographic coordinates* as a basis for locality (§2.2.2). Every replica knows its latitude and longitude, which already provides some information about locality before any network measurement. Then, in the background, OASIS estimates the geographic coordinates of every netblock on the Internet. Because the physical location of IP prefixes rarely changes [36], an accurately pinpointed network can be safely re-probed very infrequently (say, once a week). Such infrequent, background probing both reduces the risk of abuse complaints and allows fast replies to anycast requests with no need for on-demand probing.

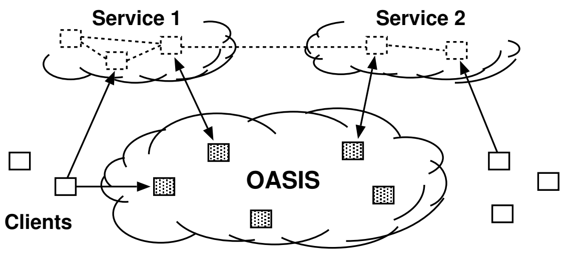

Figure 2: OASIS system overview

2.1 System overview

Figure 2 shows OASIS's high-level architecture. The system consists of a network of *core* nodes that help *clients* select appropriate *replicas* of various services. All services employ the same core nodes; we intend this set of infrastructure nodes to be small enough and sufficiently reliable so that every core node can know most of the others. Replicas also run OASIS-specific code, both to report their own load and liveness information to the core, and to assist the core with network measurements. Clients need not run any special code to use OASIS, because the core nodes provide DNS- and HTTP-based redirection services. An RPC interface is also available to OASIS-aware clients.

Though the three roles of core node, client, and replica are distinct, the same physical host often plays multiple roles. In particular, core nodes are all replicas of the OASIS RPC service, and often of the DNS and HTTP redirection services as well. Thus, replicas and clients typically use OASIS itself to find a nearby core node.

Figure 3 shows various ways in which clients and services can use OASIS. The top diagram shows an OASIS-aware client, which uses DNS-redirection to select a nearby replica of the OASIS RPC service (*i.e.*, a core node), then queries that node to determine the best replica of Service 1.

The middle diagram shows how to make legacy clients select replicas using DNS redirection. The service provider advertises a domain name served by OASIS. When a client looks up that domain name, OASIS first redirects the client's resolver to a nearby replica of the DNS service (which the resolver will cache for future accesses). The nearby DNS server then returns the address of a Service 2 replica suitable for the client. This result can be accurate if clients are near their resolvers, which is often the case [24].

The bottom diagram shows a third technique, based on service-level (*e.g.*, HTTP) redirection. Here the replicas of Service 3 are also clients of the OASIS RPC service. Each replica connects to a nearby OASIS core node selected by DNS redirection. When a client connects to a replica, that replica queries OASIS to find a better replica, then redirects the client. Such an approach does not require that clients be located near their resolvers in order to achieve high accuracy.

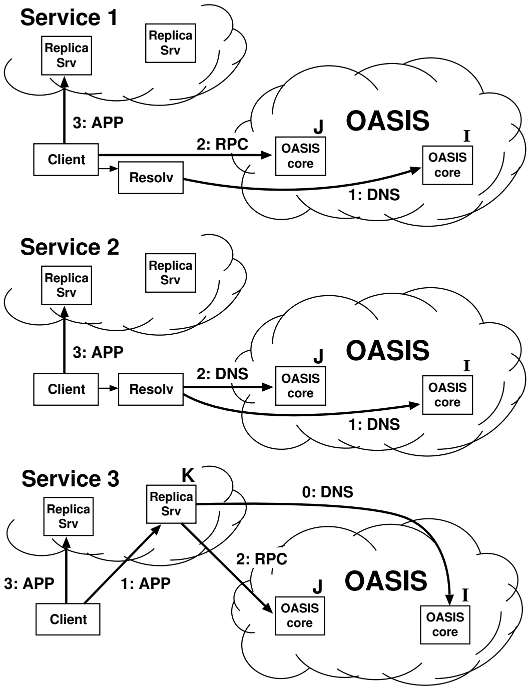

Figure 3: Various methods of using OASIS via its DNS or RPC interfaces, and the steps involved in each anycast request.

This paper largely focuses on DNS redirection, since it is the easiest to integrate with existing applications.

2.2 Design decisions

Given a client IP address and service name, the primary function of the OASIS core is to return a suitable service replica. For example, an OASIS nameserver calls its core node with the client resolver's IP address and a service name extracted from the requested domain name (*e.g.*, *coralcdn.nyuld.net* indicates service *coralcdn*).

Figure 4 shows how OASIS resolves an anycast request. First, a core node maps the client IP address to a *network bucket*, which aggregates adjacent IP addresses into netblocks of co-located hosts. It then attempts to map the bucket to a *location* (*i.e.*, coordinates). If successful, OASIS returns the closest service replica to that location (unless load-balancing requires otherwise, as described in §3.4). Otherwise, if it cannot determine the client's location, it returns a random replica.

The anycast process relies on four databases maintained in a distributed manner by the core: (1) a *service table* lists all services using OASIS (and records policy information for each service), (2) a *bucketing table* maps IP addresses to buckets, (3) a *proximity table* maps buckets to locations, and (4) one *liveness table per service* in-

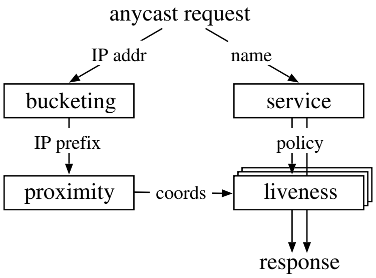

Figure 4: Logical steps to answer an anycast request

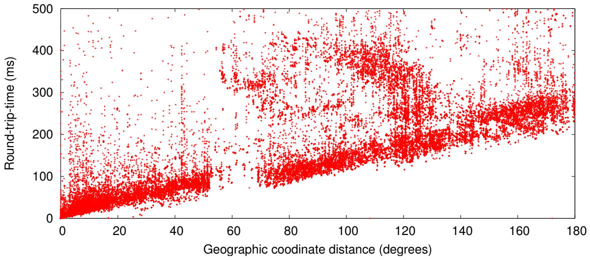

Figure 5: Correlation between round-trip-times and geographic distance across all PlanetLab hosts [43].

cludes all live replicas belonging to the service and their corresponding information (*e.g.*, coordinates, load, and capacity).

2.2.1 Buckets: The granularity of mapping hosts

OASIS must balance the precision of identifying a client's network location with its state requirements. One strawman solution is simply to probe every IP address ever seen and cache results for future requests. Many services have too large a client population for such an approach to be attractive. For DNS redirection, probing each DNS resolver would be practical if the total number of resolvers were small and constant. Unfortunately, measurements at DNS root servers [23] have shown many resolvers use dynamically-assigned addresses, thus precluding a small working set.

Fortunately, our previous research has shown that IP aggregation by prefix often preserves locality [12]. For example, more than 99% of /24 IP prefixes announced by stub autonomous systems (and 97% of /24 prefixes announced by all autonomous systems) are at the same location. Thus, we aggregate IP addresses using IP prefixes as advertised by BGP, using BGP dumps from RouteViews [38] as a starting point.[1]

However, some IP prefixes (especially larger prefixes) do not preserve locality [12]. OASIS discovers and adapts to these cases by splitting prefixes that exhibit poor locality precision,[2] an idea originally proposed by IP2Geo [30]. Using IP prefixes as network buckets not only improves scalability by reducing probing and state requirements, but also provides a concrete set of targets to *precompute*, and hence avoid on-demand probing.

2.2.2 Geographic coordinates for location

OASIS takes a two-pronged approach to locate IP prefixes: We first use a direct probing mechanism [46] to find the replica closest to the prefix, regardless of service. Then, we represent the prefix by the geographic coordinates of this closest replica and its measured round-trip-time to the prefix. We assume that all replicas know their latitude and longitude, which can easily be obtained from a variety of online services [13]. Note that OASIS's shared infrastructure design helps increase the number of vantage points and thus improves its likelihood of having a replica near the prefix.

While geographic coordinates are certainly not optimal predictors of round-trip-times, they work well in practice: The heavy band in Figure 5 shows a strong linear correlation between geographic distance and RTT. In fact, anycast only has the weaker requirement of predicting a relative ordering of nodes for a prefix, not an accurate RTT estimation. For comparison, we also implemented Vivaldi [6] and GNP [28] coordinates within OASIS; §5 includes some comparison results.

Time- and service-invariant coordinates. Since geographic coordinates are stable over time, they allow OASIS to probe each prefix infrequently. Since geographic coordinates are independent of the services, they can be shared across services—an important requirement since OASIS is designed as a shared infrastructure. Geographic coordinates remain valid even if the closest replica fails. In contrast, virtual coordinate systems [6, 28] fall short of providing either accuracy or stability [40, 46]. Similarly, simply recording a prefix's nearest replica—without its corresponding geographic coordinates—is useless if that nearest replica fails. Such an approach also requires a separate mapping per service.

Absolute error predictor. Another advantage of our two-pronged approach is that the RTT between a prefix and its closest replica is an *absolute* bound on the accuracy of the prefix's estimated location. This bound suggests a useful heuristic for deciding when to re-probe a prefix to find a better replica. If the RTT is small (a few milliseconds), reprobing is likely to have little effect. Conversely, reprobing prefixes having high RTTs to their closest replica can help improve accuracy when

[1] For completeness, we also note that OASIS currently supports aggregating by the less-locality-preserving autonomous system number, although we do not present the corresponding results in this paper.

[2] We deem that a prefix exhibits poor locality if probing different IP addresses within the prefix yields coordinates with high variance.

previous attempts missed the best replica or newly-joined replicas are closer to the prefix. Furthermore, a prefix's geographic coordinates will not change unless it is probed by a closer replica. Of course, IP prefixes can physically move, but this happens rarely enough [36] that OASIS only expires coordinates after one week. Moving a network can therefore result in sub-optimal predictions for at most one week.

Sanity checking. A number of network peculiarities can cause incorrect network measurements. For example, a replica behind a transparent web proxy may erroneously measure a short RTT to some IP prefix, when in fact it has only connected to the proxy. Replicas behind firewalls may believe they are pinging a remote network's firewall, when really they are probing their own. OASIS employs a number of tests to detect such situations (see §6). As a final safeguard, however, the core only accepts a prefix-to-coordinate mapping after seeing two consistent measurements from replicas on different networks.

In hindsight, another benefit of geographic coordinates is the ability to couple them with real-time visualization of the network [29], which has helped us identify, debug, and subsequently handle various network peculiarities.

2.2.3 System management and data replication

To achieve scalability and robustness, the location information of prefixes must be made available to all core nodes. We now describe OASIS's main system management and data organization techniques.

Global membership view. Every OASIS core node maintains a weakly-consistent view of all other nodes in the core, where each node is identified by its IP address, a globally-unique node identifier, and an incarnation number. To avoid $O(n^2)$ probing (where n is the network size), core nodes detect and share failure information cooperatively: every core node probes a random neighbor each time period (3 seconds) and, if it fails to receive a response, gossips its suspicion of failure.

Two techniques suggested by SWIM [7] reduce false failure announcements. First, several intermediates are chosen to probe this target before the initiator announces its suspicion of failure. Intermediaries alleviate the problems caused by non-transitive Internet routing [11]. Second, incarnation numbers help disambiguate failure messages: *alive* messages for incarnation i override anything for $j < i$; *suspect* for i overrides anything for $j \leq i$. If a node learns that it is suspected of failure, it increments its incarnation number and gossips its new number as alive. A node will only conclude that another node with incarnation i is dead if it has not received a corresponding alive message for $j > i$ after some time (3 minutes). This approach provides live nodes with sufficient time to respond to and correct false suspicions of failure.

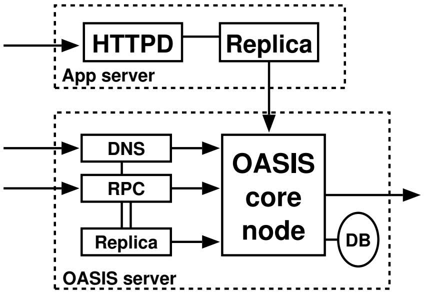

Figure 6: OASIS system components

Implicit in this design is the assumption that nodes are relatively stable; otherwise, the system would incur a high bandwidth cost for failure announcements. Given that OASIS is designed as an *infrastructure service*—to be deployed either by one service provider or a small number of cooperating providers—we believe that this assumption is reasonable.

Consistent hashing. OASIS tasks must be assigned to nodes in some globally-known yet fully-decentralized manner. For example, to decide the responsibility of mapping specific IP prefixes, we partition the set of prefixes over all nodes. Similarly, we assign specific nodes to play the role of a *service rendezvous* to aggregate information about a particular service (described in §3.3).

OASIS provides this assignment through consistent hashing [20]. Each node has a random identifier; several nodes with identifiers closest to a key—*e.g.*, the SHA-1 hash of the IP prefix or service name—in the identifier space are assigned the corresponding task. Finding these nodes is easy since all nodes have a global view. While nodes' views of the set of closest nodes are not guaranteed to be consistent, views can be easily reconciled using nodes' incarnation numbers.

Gossiping. OASIS uses gossiping to efficiently disseminate messages—about node failures, service policies, prefix coordinates—throughout the network [7]. Each node maintains a buffer of messages to be piggybacked on other system messages to *random* nodes. Each node gossips each message $O(\log n)$ times for n-node networks; such an epidemic algorithm propagates a message to all nodes in logarithmic time with high probability.[3]

Soft-state replica registration. OASIS must know all replicas belonging to a service in order to answer corresponding anycast requests. To tolerate replica failures robustly, replica information is maintained using soft-state:

[3] While structured gossiping based on consistent hashing could reduce the bandwidth overhead needed to disseminate a message [3], we use a randomized epidemic scheme for simplicity.

replicas periodically send registration messages to core nodes (currently, every 60 seconds).

Hosts running services that use OASIS for anycast—such as the web server shown in Figure 6—run a separate replica process that connects to their local application (*i.e.*, the web server) every keepalive period (currently set to 15 seconds). The application responds with its current load and capacity. While the local application remains alive, the replica continues to refresh its locality, load, and capacity with its OASIS core node.

Closest-node discovery. OASIS offloads all measurement costs to service replicas. All replicas, belonging to different services, form a lightweight overlay, in order to answer closest-replica queries from core nodes. Each replica organizes its neighbors into concentric rings of exponentially-increasing radii, as proposed by Meridian [46]: A replica accepts a neighbor for ring i only if its RTT is between 2^i and 2^{i+1} milliseconds. To find the closest replica to a destination d, a query operates in successive steps that "zero in" on the closest node in an expected $O(\log n)$ steps. At each step, a replica with RTT r from d chooses neighbors to probe d, restricting its selection to those with RTTs (to itself) between $\frac{1}{2}r$ and $\frac{3}{2}r$. The replica continues the search on its neighbor returning the minimum RTT to d. The search stops when the latest replica knows of no other potentially-closer nodes.

Our implementation differs from [46] in that we perform closest routing iteratively, as opposed to recursively: The first replica in a query initiates each progressive search step. This design trades overlay routing speed for greater robustness to packet loss.

3 Architecture

In this section, we describe the distributed architecture of OASIS in more detail: its distributed management and collection of data, locality and load optimizations, scalability, and security properties.

3.1 Managing information

We now describe how OASIS manages the four tables described in §2.2. OASIS optimizes response time by heavily replicating most information. Service, bucketing, and proximity information need only be weakly consistent; stale information only affects system performance, not its correctness. On the other hand, replica liveness information must be more fresh.

Service table. When a service initially registers with OASIS, it includes a service policy that specifies its service name and any domain name aliases, its desired server-selection algorithm, a public signature key, the maximum and minimum number of addresses to be included in responses, and the TTLs of these responses. Each core node maintains a local copy of the service table to be able to efficiently handle requests. When a new service joins OASIS or updates its existing policy, its policy is disseminated throughout the system by gossiping.

The server-selection algorithm specifies how to order replicas as a function of their distance, load, and total capacity when answering anycast requests. By default, OASIS ranks nodes by their coordinate distance to the target, favoring nodes with excess capacity to break ties. The optional signature key is used to authorize replicas registering with an OASIS core node as belonging to the service (see §3.5).

Bucketing table. An OASIS core node uses its bucketing table to map IP addresses to IP prefixes. We bootstrap the table using BGP feeds from RouteViews [38], which has approximately 200,000 prefixes. A PATRICIA trie [27] efficiently maps IP addresses to prefixes using longest-prefix matching.

When core nodes modify their bucketing table by splitting or merging prefixes [30], these changes are gossiped in order to keep nodes' tables weakly consistent. Again, stale information does not affect system correctness: prefix withdrawals are only used to reduce system state, while announcements are used only to identify more precise coordinates for a prefix.

Proximity table. When populating the proximity table, OASIS seeks to find accurate coordinates for every IP prefix, while preventing unnecessary reprobing.

OASIS maps an IP prefix to the coordinates of its closest replica. To discover the closest replica, an core node first selects an IP address from within the prefix and issues a probing request to a known replica (or first queries a neighbor to discover one). The selected replica traceroutes the requested IP to find the last routable IP address, performs closest-node discovery using the replica overlay (see §2.2.3), and, finally, returns the coordinates of the nearest replica and its RTT distance from the target IP. If the prefix's previously recorded coordinate has either expired or has a larger RTT from the prefix, the OASIS core node reassigns the prefix to these new coordinates and starts gossiping this information.

To prevent many nodes from probing the same IP prefix, the system assigns prefixes to nodes using consistent hashing. That is, several nodes closest to *hash(prefix)* are responsible for probing the prefix (three by default). All nodes go through their subset of assigned prefixes in random order, probing the prefix if its coordinates have not been updated within the last T_p seconds. T_p is a function of the coordinate's error, such that highly-accurate coordinates are probed at a slower rate (see §2.2.2).

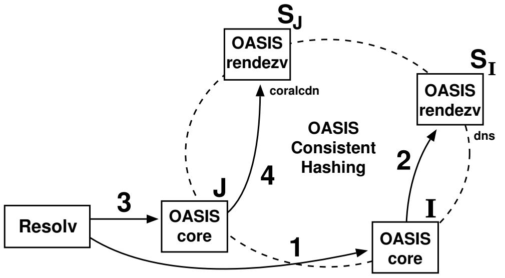

Figure 7: Steps involved in a DNS anycast request to OASIS using rendezvous nodes.

Liveness table. For each registered service, OASIS maintains a liveness table of known replicas. Gossiping is not appropriate to maintain these liveness tables at each node: stale information could cause nodes to return addresses of failed replicas, while high replica churn would require excessive gossiping and hence bandwidth consumption.

Instead, OASIS aggregates liveness information about a particular service at a few *service rendezvous* nodes, which are selected by consistent hashing. When a replica joins or leaves the system, or undergoes a significant load change, the OASIS core node with which it has registered sends an update to one of the k nodes closest to *hash(service)*. For scalability, these rendezvous nodes only receive occasional state updates, not each soft-state refresh continually sent by replicas to their core nodes. Rendezvous nodes can dynamically adapt the parameter k based on load, which is then gossiped as part of the service's policy. By default, $k = 4$, which is also fixed as a lower bound.

Rendezvous nodes regularly exchange liveness information with one another, to ensure that their liveness tables remain weakly consistent. If a rendezvous node detects that an core node fails (via OASIS's failure detection mechanism), it invalidates all replicas registered by that node. These replicas will subsequently re-register with a different core node and their information will be re-populated at the rendezvous nodes.

Compared to logically-decentralized systems such as DHTs [39, 42], this aggregation at rendezvous nodes allows OASIS to provide faster response (similar to one-hop lookups) and to support complex anycast queries (*e.g.*, as a function of both locality and load).

3.2 Putting it together: Resolving anycast

Given the architecture that we have presented, we now describe the steps involved when resolving an anycast request (see Figure 7). For simplicity, we limit our discussion to DNS redirection. When a client queries OASIS for the hostname *coralcdn.nyuld.net* for the first time:

1. The client queries the DNS root servers, finding an OASIS nameserver I for *nyuld.net* to which it sends the request.
2. **Core lookup:** OASIS core node I finds other core nodes near the client that support the DNS interface by executing the following steps:
   (a) I locally maps the client's IP address to IP prefix, and then prefix to location coordinates.
   (b) I queries one of the k rendezvous nodes for service *dns*, call this node S_I, sending the client's coordinates.
   (c) S_I responds with the best-suited OASIS nameservers for the specified coordinates.
   (d) I returns this set of DNS replicas to the client. Let this set include node J.
3. The client resends the anycast request to J.
4. **Replica lookup:** Core node J finds replicas near the client using the following steps:
   (a) J extracts the request's service name and maps the client's IP address to coordinates.
   (b) J queries one of the k rendezvous nodes for service *coralcdn*, call this S_J.
   (c) S_J responds with the best *coralcdn* replicas, which J returns to the client.

Although DNS is a stateless protocol, we can force legacy clients to perform such two-stage lookups, as well as signal to their nameservers which stage they are currently executing. §4 gives implementation details.

3.3 Improving scalability and latency

While OASIS can support a large number of replicas by simply adding more nodes, the anycast protocol described in §3.2 has a bottleneck in scaling to large numbers of clients for a particular service: one of the k rendezvous nodes is involved in each request. We now describe how OASIS reduces these remote queries to improve both scalability and client latency.

Improving core lookups. OASIS first reduces load on rendezvous nodes by lowering the frequency of core lookups. For DNS-based requests, OASIS uses relatively-long TTLs for OASIS nameservers (currently 15 minutes) compared to those for third-party replicas (configurable per service, 60 seconds by default). These longer TTLs seem acceptable given that OASIS is an infrastructure service, and that resolvers can failover between nameservers since OASIS returns multiple, geo-diverse nameservers.

Second, we observe that core lookups are rarely issued to *random* nodes: Core lookups in DNS will initially go to one of the twelve primary nameservers registered for *.nyuld.net* in the main DNS hierarchy. So, we can arrange the OASIS core so that these 12 primary nameservers play the role of rendezvous nodes for *dns*, by simply having them choose $k = 12$ consecutive node identifiers for consistent hashing (in addition to their normal random identifiers). This configuration reduces latency by avoiding remote lookups.

Improving replica lookups. OASIS further reduces load by leveraging request locality. Since both clients and replicas are redirected to their nearest OASIS core nodes—when performing anycast requests and initiating registration, respectively—hosts redirected to the same core node are likely to be close to one another. Hence, on receiving a replica lookup, an core node first checks its local liveness table for any replica that satisfies the service request.

To improve the effectiveness of using local information, OASIS also uses *local flooding*: Each core node receiving registrations sends these local replica registrations to some of its closest neighbors. ("Closeness" is again calculated using coordinate distance, to mirror the same selection criterion used for anycast.) Intuitively, this approach helps prevent situations in which replicas and clients select different co-located nodes and therefore lose the benefit of local information. We analyze the performance benefit of local flooding in §5.1.

OASIS implements other obvious strategies to reduce load, including having core nodes cache replica information returned by rendezvous nodes and batch replica updates to rendezvous nodes. We do not discuss these further due to space limitations.

3.4 Selecting replicas based on load

While our discussion has mostly focused on locality-based replica selection, OASIS supports multiple selection algorithms incorporating factors such as load and capacity. However, in most practical cases, load-balancing need not be perfect; a reasonably good node is often acceptable. For example, to reduce costs associated with "95th-percentile billing," only the elimination of traffic spikes is critical. To eliminate such spikes, a service's replicas can track their 95% bandwidth usage over five-minute windows, then report their load to OASIS as the logarithm of this bandwidth usage. By specifying load-based selection in its policy, a service can ensure that its 95% bandwidth usage at its most-loaded replica is within a factor of two of its least-loaded replica; we have evaluated this policy in §5.2.

However, purely load-based metrics cannot be used in conjunction with many of the optimizations that reduce replica lookups to rendezvous nodes (§3.3), as locality does not play a role in such replica selection. On the other hand, the computation performed by rendezvous nodes when responding to such replica lookups is much lower: while answering locality-based lookups requires the rendezvous node to compute the closest replica(s) with respect to the client's location, answering load-based lookups requires the node simply to return the first element(s) of a single list of service replicas, sorted by increasing load. The ordering of this list needs to be recomputed only when replicas' loads change.

3.5 Security properties

OASIS has the following security requirements. First, it should prohibit unauthorized replicas from joining a registered service. Second, it should limit the extent to which a particular service's replicas can inject bad coordinates. Finally, it should prevent adversaries from using the infrastructure as a platform for DDoS attacks.

We assume that all OASIS core nodes are trusted; they do not gossip false bucketing, coordinates, or liveness information. We also assume that core nodes have loosely synchronized clocks to verify expiry times for replicas' authorization certificates. (Loosely-synchronized clocks are also required to compare registration expiry times in liveness tables, as well as measurement times when determining whether to reprobe prefixes.) Additionally, we assume that services joining OASIS have some secure method to initially register a public key. An infrastructure deployment of OASIS may have a single or small number of entities performing such admission control; the service provider(s) deploying OASIS's primary DNS nameservers are an obvious choice. Less secure schemes such as using DNS TXT records may also be appropriate in certain contexts.

To prevent unauthorized replicas from joining a service, a replica must present a valid, fresh certificate signed by the service's public key when initially registering with the system. This certificate includes the replica's IP address and its coordinates. By providing such admission control, OASIS only returns IP addresses that are authorized as valid replicas for a particular service.

OASIS limits the extent to which replicas can inject bad coordinates by evicting faulty replicas or their corresponding services. We believe that sanity-checking coordinates returned by the replicas—coupled with the penalty of eviction—is sufficient to deter services from assigning inaccurate coordinates for their replicas and replicas from responding falsely to closest-replica queries from OASIS.

Finally, OASIS prevents adversaries from using it as a platform for distributed denial-of-service attacks by requiring that replicas accept closest-replica requests only from core nodes. It also requires that a replica's overlay neighbors are authorized by OASIS (hence, replicas

```
;; ANSWER SECTION:
example.net.nyud.net                       600 IN CNAME
             coralcdn.ab4040d9a9e53205.oasis.nyuld.net.

coralcdn.ab4040d9a9e53205.oasis.nyuld.net.  60 IN A
                                          171.64.64.217

;; AUTHORITY SECTION:
ab4040d9a9e53205.oasis.nyuld.net.          600 IN NS
                     171.64.64.217.ip4.oasis.nyuld.net.
ab4040d9a9e53205.oasis.nyuld.net.          600 IN NS
                      169.229.50.5.ip4.oasis.nyuld.net.
```

Figure 8: Output of `dig` for a hostname using OASIS.

only accept probing requests from other approved replicas). OASIS itself has good resistance to DoS attacks, as most client requests can be resolved using information stored locally, *i.e.*, not requiring wide-area lookups between core nodes.

4 Implementation

OASIS's implementation consists of three main components: the OASIS core node, the service replica, and stand-alone interfaces (including DNS, HTTP, and RPC). All components are implemented in C++ and use the asynchronous I/O library from the SFS toolkit [25], structured using asynchronous events and callbacks. The core node comprises about 12,000 lines of code, the replica about 4,000 lines, and the various interfaces about 5,000 lines. The bucketing table is maintained using an in-memory PATRICIA trie [27], while the proximity table uses BerkeleyDB [41] for persistent storage.

OASIS's design uses static latitude/longitude coordinates with Meridian overlay probing [46]. For comparison purposes, OASIS also can be configured to use synthetic coordinates using Vivaldi [6] or GNP [28].

RPC and HTTP interfaces. These interfaces take an optional target IP address as input, as opposed to simply using the client's address, in order to support integration of third-party services such as HTTP redirectors (Figure 3). Beyond satisfying normal anycast requests, these interfaces also enable a localization service by simply exposing OASIS's proximity table, so that any client can ask "What are the coordinates of IP *x*?"[4] In addition to HTML, the HTTP interface supports XML-formatted output for easy visualization using online mapping services [13].

DNS interface. OASIS takes advantage of low-level DNS details to implement anycast. First, a nameserver must differentiate between core and replica lookups. Core lookups only return *nameserver* (NS) records for nearby OASIS nameservers. Replica lookups, on the other hand, return *address* (A) records for nearby replicas. Since DNS is a stateless protocol, we signal the type of a client's request in its DNS query: replica lookups all have *oasis* prepended to *nyuld.net*. We force such signalling by returning CNAME records during core lookups, which map aliases to their *canonical names*.

This technique alone is insufficient to force many client resolvers, including BIND, to immediately issue replica lookups to these nearby nameservers. We illustrate this with an example query for CoralCDN [10], which uses the service alias $*.nyud.net$. A resolver R discovers nameservers u,v for $nyud.net$ by querying the root servers for $example.net.nyud.net$.[5] Next, R queries u for this hostname, and is returned a CNAME for $example.net.nyud.net \to coralcdn.oasis.nyuld.net$ and NS x,y for $coralcdn.oasis.nyuld.net$. In practice, R will reissue a new query for $coralcdn.oasis.nyuld.net$ to nameserver v, which is not guaranteed to be close to R (and v's local cache may include replicas far from R).

We again use the DNS query string to signal whether a client is contacting the correct nameservers. When responding to core lookups, we encode the set of NS records in hex format (ab4040d9a9e53205) in the returned CNAME record (Figure 8). Thus, when v receives a replica lookup, it checks whether the query encodes its own IP address, and if it does not, immediately re-returns NS records for x,y. Now, having received NS records authoritative for the name queried, a resolver contacts the desired nameservers x or y, which returns an appropriate replica for *coralcdn*.

5 Evaluation

We evaluate OASIS's performance benefits for DNS-based anycast, as well as its scalability and bandwidth trade-offs.

5.1 Wide-area evaluation of OASIS

Experimental setup. We present wide-area measurements on PlanetLab [34] that evaluate the accuracy of replica selection based on round-trip-time and throughput, DNS response time, and the end-to-end time for a simulated web session. In all experiments, we ran replicas for one service on approximately 250 PlanetLab hosts spread around the world (including 22 in Asia), and we ran core nodes and DNS servers on 37 hosts.[6]

[4] We plan to support such functionality with DNS TXT records as well, although this has not been implemented yet.

[5] To adopt OASIS yet preserve its own top-level domain name, CoralCDN points the NS records for $nyud.net$ to OASIS's nameservers; $nyud.net$ is registered as an alias for *coralcdn* in its service policy.

[6] This number was due to the unavailability of UDP port 53 on most PlanetLab hosts, especially given CoralCDN's current use of same.

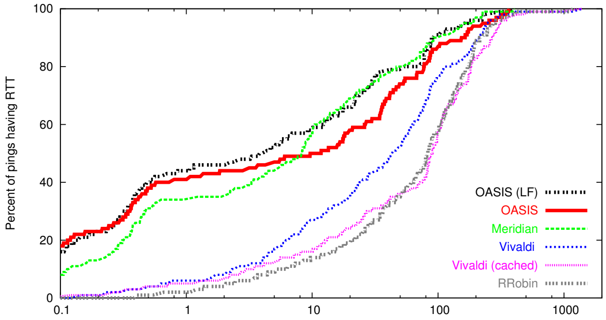

Figure 9: Round trip times (ms)

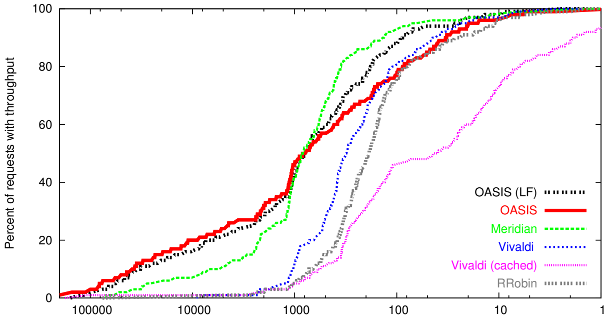

Figure 10: Client-server TCP throughput (KB/s)

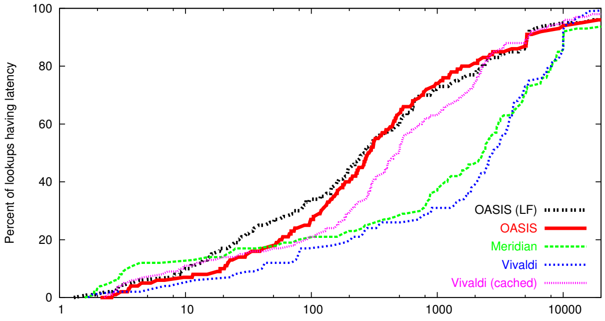

Figure 11: DNS resolution time (ms) for new clients

We compare the performance of replica selection using six different anycast strategies: (1) *OASIS (LF)* refers to the OASIS system, using both local caching and local flooding (to the nearest three neighbors; see §3.3). (2) *OASIS* uses only local caching for replicas. (3) *Meridian* (our implementation of [46]) performs on-demand probing by executing closest-replica discovery whenever it receives a request. (4) *Vivaldi* uses 2-dimensional dynamic virtual coordinates [6], instead of static geographic coordinates, by probing the client from 8-12 replicas on-demand. The core node subsequently computes the client's virtual coordinates and selects its closest replica based on virtual coordinate distance. (5) *Vivaldi (cached)* probes IP prefixes in the background, instead of on-demand. Thus, it is similar to OASIS with local caching, except for using virtual coordinates to populate OASIS's proximity table. (6) Finally, *RRobin* performs round-robin DNS redirection amongst all replicas in the system, using a single DNS server located at Stanford University.

We performed client measurements on the same hosts running replicas. However, we configured OASIS so that when a replica registers with an OASIS core node, the node does *not* directly save a mapping from the replica's prefix to its coordinates, as OASIS would do normally. Instead, we rely purely on OASIS's background probing to assign coordinates to the replica's prefix.

Three consecutive experiments were run at each site when evaluating ping, DNS, and end-to-end latencies. Short DNS TTLs were chosen to ensure that clients contacted OASIS for each request. Data from all three experiments are included in the following cumulative distribution function (CDF) graphs.

Minimizing RTTs. Figures 9 shows the CDFs of round-trip-times in log-scale between clients and their returned replicas. We measured RTTs via ICMP echo messages, using the ICMP response's kernel timestamp when calculating RTTs. RTTs as reported are the minimum of ten consecutive probes. We see that OASIS and Meridian significantly outperform anycast using Vivaldi and round robin by one to two orders of magnitude.

Two other interesting results merit mention. First, *Vivaldi (cached)* performs significantly worse than on-demand *Vivaldi* and even often worse than *RRobin*. This arises from the fact that *Vivaldi* is not stable over time with respect to coordinate translation and rotation. Hence, cached results quickly become inaccurate, although recent work has sought to minimize this instability [8, 33]. Second, OASIS outperforms *Meridian* for 60% of measurements, a rather surprising result given that OASIS *uses* Meridian as its background probing mechanism. It is here where we see OASIS's benefit from using RTT as an absolute error predictor for coordinates (§2.2.2): reprobing by OASIS yields strictly better results, while the accuracy of Meridian queries can vary.

Maximizing throughput. Figure 10 shows the CDFs of the steady-state throughput from replicas to their clients, to examine the benefit of using nearby servers to improve data-transfer rates. TCP throughput is measured using `iperf-1.7.0` [18] in its default configuration (a TCP window size of 32 KB). The graph shows TCP performance in steady-state. OASIS is competitive with or superior to all other tested systems, demonstrating its performance for large data transfers.

DNS resolution time. Figures 11 and 12 evaluate the DNS performance for new clients and for clients already caching their nearby OASIS nameservers, respectively. A request by a new client includes the time to

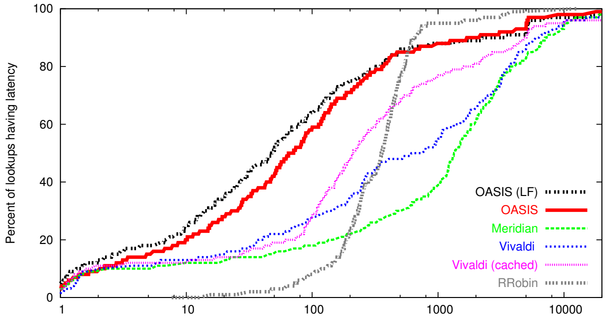

Figure 12: DNS resolution time (ms) for replica lookups

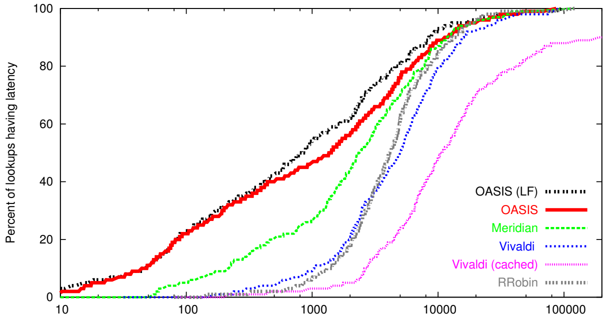

Figure 13: End-to-end download performance (ms)

perform three steps: (1) contact an initial OASIS core node to learn a nearby nameserver, (2) re-contact a distant node and again receive NS records for the same nearby nameservers (see §4), and (3) contact a nearby core node as part of a replica lookup. Note that we did not specially configure the 12 primary nameservers as rendezvous nodes for *dns* (see §3.3), and thus use a wide-area lookup during Step 1. This two-step approach is taken by all systems: *Meridian* and *Vivaldi* both perform on-demand probing twice. We omit *RRobin* from this experiment, however, as it always uses a single nameserver. Clients already caching nameserver information need only perform Step 3, as given in Figure 12.

OASIS's strategy of first finding nearby nameservers and then using locally-cached information can achieve significantly faster DNS response times compared to on-demand probing systems. The median DNS resolution time for OASIS replica lookups is almost 30*x* faster than that for *Meridian*.[7] We also see that local flooding can improve median performance by 40% by reducing the number of wide-area requests to rendezvous nodes.

End-to-end latency. Figure 13 shows the end-to-end time for a client to perform a synthetic web session, which includes first issuing a replica lookup via DNS and then downloading eight 10KB files sequentially. This file size is chosen to mimic that of common image files, which are often embedded multiple times on a given web page. We do not simulate persistent connections for our transfers, so each request establishes a new TCP connection before downloading the file. Also, our faux-webserver never touches the disk, so does not take (PlanetLab's high) disk-scheduling latency into account.

End-to-end measurements underscore OASIS's true performance benefit, coupling very fast DNS response time with very accurate server selection. Median response-time for OASIS is 290% faster than *Meridian* and 500% faster than simple round-robin systems.

metric	california	texas	new york	germany
latency	23.3	0.0	0.0	0.0
load	9.0	11.3	9.6	9.2

Table 1: 95th-percentile bandwidth usage (MB)

5.2 Load-based replica selection

This section considers replica selection based on load. We do not seek to quantify an optimal load- and latency-aware selection metric; rather, we verify OASIS's ability to perform load-aware anycast. Specifically, we evaluate a load-balancing strategy meant to reduce costs associated with 95th-percentile billing (§3.4).

In this experiment, we use four distributed servers that run our faux-webserver. Each webserver tracks its bandwidth usage per minute, and registers its load with its local replica as the logarithm of its 95th-percentile usage. Eight clients, all located in California, each make 50 anycast requests for a 1MB file, with a 20-second delay between requests. (DNS records have a TTL of 15 seconds.)

Table 1 shows that the webserver with highest bandwidth costs is easily within a factor of two of the least-loaded server. On the other hand, locality-based replica selection creates a traffic spike at a single webserver.

5.3 Scalability

Since OASIS is designed as an infrastructure system, we now verify that a reasonable-sized OASIS core can handle Internet-scale usage.

Measurements at DNS root servers have shown steady traffic rates of around 6.5M A queries per 10 minute interval across all $\{e,i,k,m\}$*.root-servers.net* [23]. With a deployment of 1000 OASIS DNS servers—and, for simplicity, assuming an even distribution of requests to nodes—even if OASIS received requests at an equivalent rate, each node would see only 10 requests per second.

On the other hand, OASIS often uses shorter TTLs to handle replica failover and load balancing. The same datasets showed approximately 100K unique resolvers

[7] A recursive Meridian implementation [46] may be faster than our iterative implementation: our design emphasizes greater robustness to packet loss, given our preference for minimizing probing.

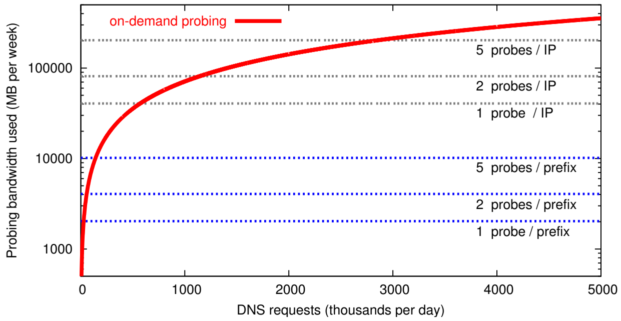

Figure 14: Bandwidth trade-off between on-demand probing, caching IP prefixes (OASIS), and caching IP addresses

per 10 minute interval. Using the default TTL of 60 seconds, even if every client re-issued a request every 60 seconds for all s services using OASIS, each core node would receive at most $1.6 \cdot s$ requests per second.

To consider one real-world service, as opposed to some upper bound for all Internet traffic, CoralCDN [10] handles about 20 million HTTP requests from more than one million unique client IPs per day (as of December 2005). To serve this web population, CoralCDN answers slightly fewer than 5 million DNS queries (for all query types) per day, using TTLs of 30-60 seconds. This translates to a system *total* of 57 DNS queries per second.

5.4 Bandwidth trade-offs

This section examines the bandwidth trade-off between precomputing prefix locality and performing on-demand probing. If a system receives only a few hundred requests per week, OASIS's approach of probing every IP prefix is not worthwhile. Figure 14 plots the amount of bandwidth used in caching and on-demand anycast systems for a system with 2000 replicas. Following the results of [46], we estimate each closest-replica query to generate about 10.4 KB of network traffic (load grows sub-linearly with the number of replicas).

Figure 14 simulates the amount of bandwidth used per week for up to 5 million DNS requests per day (the request rate from CoralCDN), where each results in a new closest-replica query. OASIS's probing of 200K prefixes—even when each prefix may be probed multiple times—generates orders of magnitude less network traffic. We also plot an upper-bound on the amount of traffic generated if the system were to cache IP addresses, as opposed to IP prefixes.

Project	Service	Description
ChunkCast [5]	chunkcast	Anycast gateways
CoralCDN [10]	coralcdn	Web proxies
Na Kika [14]	nakika	Web proxies
OASIS	dns	DNS interface
	http	HTTP interface
	rpc	RPC interface
OCALA [19]	ocala	Client IP gateways
	ocalarsp	Server IP gateways
OpenDHT [37]	opendht	Client DHT gateways

Figure 15: Services using OASIS as of March 2006. Services can be accessed using $\langle service \rangle .nyuld.net$.

While one might expect the number of DNS resolvers to be constant and relatively small, many resolvers use dynamically-assigned addresses and thus preclude a small working set: the root-servers saw more than 4 million unique clients in a week, with the number of clients increasing linearly after the first day's window [23]. Figure 14 uses this upper-bound to plot the amount of traffic needed when caching IP addresses. Of course, new IP addresses always need to be probed on-demand, with the corresponding performance hit (per Figure 12).

6 Deployment lessons

OASIS has been deployed on about 250 PlanetLab hosts since November 2005. Figure 15 lists the systems currently using OASIS and a brief description of their service replicas. We present some lessons that we learned in the process.

Make it easy to integrate. Though each application server requires a local replica, for a shared testbed such as PlanetLab, a single replica process on a host can serve on behalf of multiple local processes running different applications. To facilitate this, we now run OASIS replicas as a public service on PlanetLab; to adopt OASIS, PlanetLab applications need only listen on a registered port and respond to keepalive messages.

Applications can integrate OASIS even without any source-code changes or recompilation. Operators can run or modify simple stand-alone scripts we provide that answer replica keepalive requests after simple liveness and load checks (via `ps` and the `/proc` filesystem).

Check for proximity discrepancies. Firewalls and middleboxes can lead one to draw false conclusions from measurement results. Consider the following two problems we encountered, mentioned earlier in §2.2.2.

To determine a routable IP address in a prefix, a replica performs a traceroute and uses the last reachable node that responded to the traceroute. However, since firewalls can perform egress filtering on ICMP packets, an unsuspecting node would then ask others to probe its own egress point, which may be far away from the desired prefix. Hence, replicas initially find their immediate upstream routers—*i.e.*, the set common to multiple traceroutes—which they subsequently ignored.

When replicas probe destinations on TCP port 80 for closest-replica discovery, any local transparent web proxy will perform full TCP termination, leading an unsuspecting node to conclude that it is very close to the destination. Hence, a replica first checks for a transparent proxy, then tries alternative probing techniques.

Both problems would lead replicas to report themselves as incorrectly close to some IP prefix. So, by employing measurement redundancy, OASIS can compare answers for precision and sanity.

Be careful *what* you probe. No single probing technique is both sufficiently powerful and innocuous (from the point-of-view of intrusion-detection systems). As such, OASIS has adapted its probing strategies based on ISP feedback. ICMP probes and TCP probes to random high ports were often dropped by egress firewalls and, for the latter, flagged as unwanted port scans. Probing to TCP port 80 faced the problem of transparent web proxies, and probes to TCP port 22 were often flagged as SSH login attacks. Unfortunately, as OASIS performs probing from multiple networks, automated abuse complaints from IDSs are sent to many separate network operators. Currently, OASIS uses a mix of TCP port 80 probes, ICMP probes, and reverse DNS name queries.

Be careful *whom* you probe. IDSs deployed on some networks are incompatible with active probing, irrespective of the frequency of probes. Thus, OASIS maintains and checks a blacklist whenever a target IP prefix or address is selected for probing. We apply this blacklist at all stages of probing: Initially, only the OASIS core checked target IP prefixes. However, this strategy led to abuse complaints from ASes that provide transit for the target, yet filter ICMPs; in such cases, replicas tracerouting the prefix would end up probing the upstream AS.

7 Related work

We classify related work into two areas most relevant to OASIS: network distance estimation and server selection. Network distance estimation techniques are used to identify the location and/or distance between hosts in the network. The server-selection literature deals with finding an appropriately-located server (possibly using network distance estimation) for a client request.

Network distance estimation. Several techniques have been proposed to reduce the amount of probing per request. Some initial proposals (such as [16]) are based on the triangle-inequality assumption. IDMaps [9] proposed deploying *tracers* that all probe one another; the distance between two hosts is calculated as the sum of the distances between the hosts and their selected tracers, and between the two selected tracers. Theilmann and Rothermel described a hierarchical tree-like system [44], and Iso-bar proposed a two-tier system using landmark-based clustering algorithms [4]. King [15] used recursive queries to remote DNS nameservers to measure the RTT distance between any two *non-participating* hosts.

Recently, virtual coordinate systems (such as GNP [28] and Vivaldi [6]) offer new methods for latency estimation. Here, nodes generate synthetic coordinates after probing one another. The distance between peers in the coordinate space is used to predict their RTT, the accuracy of which depends on how effectively the Internet can be embedded into a d-dimensional (usually Euclidean) space.

Another direction for network estimation has been the use of geographic mapping techniques; the main idea is that if geographic distance is a good indicator of network distance, then estimating geographic location accurately would obtain a first approximation for the network distance between hosts. Most approaches in geographic mapping are heuristic. The most common approaches include performing queries against a `whois` database to extract city information [17, 32], or tracerouting the address space and then mapping router names to locations based on ISP-specific naming conventions [12, 30]. Commercial entities have sought to create exhaustive IP-range mappings [1, 35].

Server selection. IP anycast was proposed as a network-level solution to server selection [22, 31]. However, with various deployment and scalability problems, IP anycast is not widely used or available. Recently, PIAS has argued for supporting IP anycast as a proxy-based service to overcome deployment challenges [2]; OASIS can serve as a powerful and flexible server-selection backend for such a system.

One of the largest deployed content distribution networks, Akamai [1] reportedly traceroutes the IP address space from multiple vantage points to detect route convergence, then pings the common router from every data center hosting an Akamai cluster [4]. OASIS's task is more difficult than that of commercial CDNs, given its goal of providing anycast for multiple services.

Recent literature has proposed techniques to minimize such exhaustive probing. Meridian [46] (used for DNS redirection by [45]) creates an overlay network with neighbors chosen from a particular distribution; routing to closer nodes is guaranteed to find a minimum given a growth-restricted metric space [21]. In contrast, OASIS completely eliminates on-demand probing.

OASIS allows more flexible server selection than pure locality-based solutions, as it stores load and capacity estimates from replicas in addition to locality information.

8 Conclusion

OASIS is a global distributed anycast system that allows legacy clients to find nearby or unloaded replicas for distributed services. Two main features distinguish OASIS from prior systems. First, OASIS allows multiple application services to share the anycast service. Second, OASIS avoids on-demand probing when clients initiate requests. Measurements from a preliminary deployment show that OASIS, provides a significant improvement in the performance that clients experience over state-of-the-art on-demand probing and coordinate systems, while incurring much less network overhead.

OASIS's contributions are not merely its individual components, but also the deployed system that is immediately usable by both legacy clients and new services. Publicly deployed on PlanetLab, OASIS has already been adopted by a number of distributed services [5, 10, 14, 19, 37].

Acknowledgments. We thank A. Nicolosi for the keyword analysis of Figure 1. M. Huang, R. Huebsch, and L. Peterson have aided our efforts to run PlanetLab services. We also thank D. Andersen, S. Annapureddy, N. Feamster, J. Li, S. Rhea, I. Stoica, our anonymous reviewers, and our shepherd, S. Gribble, for comments on drafts of this paper. This work was conducted as part of Project IRIS under NSF grant ANI-0225660.

References

[1] Akamai Technologies. http://www.akamai.com/, 2006.

[2] H. Ballani and P. Francis. Towards a global IP anycast service. In *SIGCOMM*, 2005.

[3] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI*, May 2005.

[4] Y. Chen, K. H. Lim, R. H. Katz, and C. Overton. On the stability of network distance estimation. *SIGMETRICS Perform. Eval. Rev.*, 30(2):21–30, 2002.

[5] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz. ChunkCast: An anycast service for large content distribution. In *IPTPS*, Feb. 2006.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *SIGCOMM*, Aug. 2004.

[7] A. Das, I. Gupta, and A. Motivala. SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *Dependable Systems and Networks*, June 2002.

[8] C. de Launois, S. Uhlig, and O. Bonaventure. A stable and distributed network coordinate system. Technical report, Universite Catholique de Louvain, Dec. 2004.

[9] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Trans. on Networking*, Oct. 2001.

[10] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *NSDI*, Mar. 2004.

[11] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. Non-transitive connectivity and DHTs. In *WORLDS*, Dec. 2005.

[12] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan. Geographic locality of IP prefixes. In *IMC*, Oct. 2005.

[13] Google Maps. http://maps.google.com/, 2006.

[14] R. Grimm, G. Lichtman, N. Michalakis, A. Elliston, A.Kravetz, J. Miller, and S. Raza. Na Kika: Secure service execution and composition in an open edge-side computing network. In *NSDI*, May 2006.

[15] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *IMW*, 2001.

[16] J. Guyton and M. Schwartz. Locating nearby copies of replicated Internet servers. In *SIGCOMM*, Aug. 1995.

[17] IP to Lat/Long server, 2005. http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll/.

[18] Iperf. Version 1.7.0 – the TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/, 2005.

[19] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An architecture for supporting legacy applications over overlays. In *NSDI*, May 2006.

[20] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, May 1997.

[21] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *STOC*, 2002.

[22] D. Katabi and J. Wroclawski. A framework for scalable global IP-anycast (GIA). In *SIGCOMM*, Aug. 2000.

[23] K. Keys. Clients of DNS root servers, 2002-08-14. http://www.caida.org/projects/dns-analysis/, 2002.

[24] Z. M. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A precise and efficient evaluation of the proximity between web clients and their local DNS servers. In *USENIX Conference*, June 2002.

[25] D. Mazières. A toolkit for user-level file systems. In *USENIX Conference*, June 2001.

[26] A. Mislove, A. Post, A. Haeberlen, and P. Druschel. Experiences in building and operating a reliable peer-to-peer application. In *EuroSys*, Apr. 2006.

[27] D. Morrison. Practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4), Oct. 1968.

[28] E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, June 2002.

[29] OASIS. http://www.coralcdn.org/oasis/, 2006.

[30] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for Internet hosts. In *SIGCOMM*, Aug. 2001.

[31] C. Patridge, T. Mendez, and W. Milliken. Host anycasting service. RFC 1546, Network Working Group, Nov. 1993.

[32] D. M. R. Periakaruppan and J. Donohoe. Where in the world is netgeo.caida.org? In *INET*, June 2000.

[33] P. Pietzuch, J. Ledlie, and M. Seltzer. Supporting network coordinates on planetlab. In *WORLDS*, Dec. 2005.

[34] PlanetLab. http://www.planet-lab.org/, 2005.

[35] Quova. http://www.quova.com/, 2006.

[36] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *IMW*, Nov. 2002.

[37] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *SIGCOMM*, Aug. 2005.

[38] RouteViews. http://www.routeviews.org/, 2006.

[39] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, Nov 2001.

[40] K. Shanahan and M. J. Freedman. Locality prediction for oblivious clients. In *IPTPS*, Feb. 2005.

[41] Sleepycat. BerkeleyDB v4.2, 2005.

[42] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. In *IEEE/ACM Trans. on Networking*, 2002.

[43] J. Stribling. PlanetLab AllPairsPing data, 08-03-2005:11:14:19. http://www.pdos.lcs.mit.edu/ strib/pl_app/, 2005.

[44] W. Theilmann and K. Rothermel. Dynamic distance maps of the Internet. In *IEEE INFOCOM*, Mar 2001.

[45] B. Wong and E. G. Sirer. ClosestNode.com: an open access, scalable, shared geocast service for distributed systems. *SIGOPS OSR*, 40(1), 2006.

[46] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *SIGCOMM*, Aug. 2005.