Towards a global IP anycast service

Paper 110, 14 Pages

Abstract

IP anycast, with its innate ability to find nearby resources in a robust and efficient fashion, has long been considered an important means of service discovery. The growth of P2P applications presents appealing new uses for IP anycast. Unfortunately, IP anycast suffers from serious problems: it is very hard to deploy globally, it scales poorly by the number of anycast groups, and it lacks important features like load-balancing. As a result, its use is limited to a few critical infrastructure services such as DNS root-servers. The primary contribution of this paper is a new IP anycast architecture, NIRN, that overcomes these problems while largely maintaining the strengths of IP anycast. NIRN makes use of a proxy overlay that advertises IP anycast addresses on behalf of group members and tunnels anycast packets to those members. The paper presents a detailed design of NIRN, evaluates its scalability and efficiency through simulation and presents micro-benchmarks of our implementation. We also present preliminary measurement results on anycasted DNS root-servers that suggest that IP anycast provides good affinity and offer insights into deployment strategies that might enable it to attain good proximity properties. Finally, we describe how NIRN supports two important P2P and overlay applications.

I. Introduction

Ever since it was proposed in 1993, IP anycast[1] has been viewed as a powerful IP packet addressing and delivery mode. Because IP anycast typically routes packets to the nearest of a group of hosts, it has been seen as a way to obtain efficient, transparent and robust ***service discovery***. In cases where the service itself is a connectionless ***query/reply service***, IP anycast supports the complete service, not just discovery of the service. The best working example of the latter is the use of IP anycast to replicate root DNS servers [5][9] without modifying DNS clients. Other proposed uses include ***host autoconfiguration***[2] and using anycast to reach a ***routing substrate***, such as rendezvous points for a multicast tree[4][8] or a IPv6 to IPv4 (6to4) transition device[3].

In spite of its benefits, there has been very little IP anycast deployment to date, especially on a global scale. The only global scale use of IP anycast in a production environment that we are aware of is the anycasting of DNS root-servers and AS-112 servers[7][2].

The reason for this is that IP anycast has serious limitations. Foremost among these is IP anycast's poor scalability. As with IP multicast, routes for IP anycast groups cannot be aggregated—the routing infrastructure must support one route per IP anycast group. It is also very hard to deploy IP anycast globally. The network administrator must obtain an address block of adequate size (i.e. a /24), and arrange to advertise it into the BGP substrate of its upstream ISPs. Finally, the use of IP routing as the host selection mechanism means that important selection metrics such as server load cannot be used. It is important to note that while IPv6 has defined anycast as part of its addressing architecture[6], it is also afflicted by the same set of problems.

By contrast, application layer anycast provides a one-to-any service by mapping a higher-level name, such as a DNS name, into one of a group of hosts, and then informing the client of the selected host's IP address, for instance through DNS or some redirect mechanism. This approach is much easier to deploy globally, and is in some ways superior in functionality to IP anycast. For example, the fine grained control over the load across group members and the ability to incorporate other such selection criteria makes DNS-based anycast the method of choice for Content Distribution Networks (CDNs) today.

In spite of this, we believe that IP anycast has compelling advantages, and its appeal increases as overlay and P2P applications increase. First, IP anycast operates at a low level. This makes it potentially useable by, and transparent to, any application that runs over IP. It also makes IP anycast the only form of anycast suitable for low-level protocols, such as DNS. Second, it automatically discovers nearby resources, eliminating the need for complex proximity discovery mechanisms [30]. Finally, packets are delivered directly to the target destination without the need for a redirect (frequently required by application-layer anycast approaches). This saves at least one packet round trip, which can be important for short lived exchanges. It is these advantages that have led to increased use of IP anycast within the operational community, both for providing useful services (DNS root-servers), and increasingly for protecting services from unwanted packets (AS112 and DDoS sinkholes [39]).

The *primary contribution* of this paper is the detailed description of a deployment architecture for an IP anycast service that overcomes the limitations of today's "native" IP anycast while adding new features, some typically associated with application-level anycast, and some completely new. This architecture, called NIRN (Not Its Real Name), is composed as an overlay, and utilizes but does not impact the IP routing infrastructure. The fact that NIRN is an IP anycast service means that *clients* use the service completely transparently—that is, with their existing IP stacks and applications.

NIRN allows an endhost in an anycast group (anycast group member, or **anycast target**) to receive anycast packets for that group via its normal unicast address (and normal proto-

[1] IP anycast is an IP addressing and delivery mode whereby an IP packet is sent to one of a group of hosts identifi ed by the IP anycast address. Where IP unicast is one-to-one, and IP multicast is one-to-many, IP anycast is one-to-any.

[2] anycasted servers that answer PTR queries for the RFC 1918 addresses

col stack). Furthermore, the anycast target *joins* the anycast group simply by transmitting a request packet to an anycast address (again, via its unicast interface). The target may likewise *leave* the group through a request packet, or by simply becoming silent.

NIRN utilizes the IP address space efficiently: thousands of IP anycast groups may be identified through a single IP address. It scales well by the number of groups, group size and group churn with virtually no impact on the IP routing infrastructure. It provides fast failover in response to failures of both target hosts and NIRN infrastructure nodes.

NIRN can select targets based on criteria other than proximity to the sending host, notably including the ability to load balance among targets. NIRN has the unique feature that an anycast group member can also transmit packets to other members of the anycast group. This is in contrast to native IP anycast, where a group member would receive its own packet if it transmitted to the group. This feature now makes IP anycast available to P2P applications, something not possible if a host can't both send to and receive from the anycast group.

Additional contributions include:

1) Using a tier1 ISP topology map, we show that NIRN imposes an acceptable stretch on paths, and that this stretch improves with growth in the NIRN infrastructure.
2) Using measurements on existing DNS root server deployments, we show that native IP anycast is quite poor at finding the nearest server (in terms of RTT). We argue that this is a result of poor server placement, rather than a fundamental characteristic of native IP anycast, and suggest improvements that should fix the problem.
3) We evaluate the scalability of NIRN with large, dynamic groups through simulation.
4) We provide micro-benchmarks of our implementation.
5) Using measurements on existing DNS root server anycast deployments, we also show that IP anycast provides adequate connection affinity.

The remainder of the paper is organized as follows: Section II identifies the features of an ideal anycast service. SectionIII spells out the system design together with the goals satisfied by each design feature. Section IV uses simulations and measurements to evaluate various features of the NIRN design. It also presents details and microbenchmarks for our NIRN prototype implementation. Section V discusses related work, section VI describes a few applications made possible by NIRN. Section VII describes other important goals that NIRN must fulfill and section VIII presents our conclusions.

II. Design Goals

This section specifically lays out the design goals of NIRN, and briefly comments on how well NIRN meets those goals. The subsequent design description section refers back to these goals as appropriate. The goals are listed here in two parts. The first part lists those goals that are accomplished by native IP anycast, and that we wish to retain. The second part lists those goals that are not accomplished by native IP anycast. In this way, we effectively highlight the weaknesses of IP anycast, and the contributions of NIRN.

1) *Backwards Compatible*: Native IP anycast is completely transparent to client hosts, and we believe that this transparency is critical to the success of a new IP anycast service. Also, IP anycast operates without any changes to routers by virtue of the fact that Internet routing protocols cannot distinguish between multiple paths to the same host, or multiple paths to multiple different hosts with the same (anycast) address. Because NIRN is an overlay technology, it does not change routers.
2) *Scale by group size*: By virtue of being totally distributed among routers, native IP anycast scales well by group size. NIRN has no inherent group size limitation. NIRN is deployed as an overlay infrastructure, and can scale arbitrarily according to the size of that infrastructure.
3) *Efficient packet transfer*: Because native IP anycast uses IP routing, its paths are naturally efficient. As an overlay, NIRN imposes some stretch penalty on the paths packets take. The penalty imposed by NIRN is small (section IV-C), and shrinks as the NIRN infrastructure grows.
4) *Robustness*: Native IP anycast's robustness properties (including packet loss) are similar to IP unicast. NIRN is engineered to be similarly robust.
5) *Fast failover*: Failover speed in Native IP anycast depends on the convergence speed of the underlying routing algorithms, and can be fast (OSPF) or somewhat slow (BGP). NIRN can be engineered to almost always rely on OSPF for certain types of failover (section III-F.2). The NIRN overlay exposes additional failover situations that go beyond IP routing, and these are handled accordingly (Section III-F).

The following are the goals that native IP anycast does not satisfy.

6) *Ease of* joining *and* leaving: Target hosts must not have to interact with routing to join and leave.
7) *Scale by the number of groups*: In addition to scaling by the usual metrics of memory and bandwidth, we require that NIRN also make efficient use of the IP address space. NIRN is able to accommodate thousands of groups within a single address by incorporating TCP and UDP port numbers as part of the group address.
8) *Scale by group dynamics*: Globally, IP routing behaves very badly when routes are frequently added and withdrawn. The NIRN overlay hides member dynamics from IP routing, and can handle dynamics caused both by continuous member churn and flash crowds (including those caused by DDoS attacks).
9) *Target Selection criteria*: IP anycast can only select targets based on proximity. At a minimum, we wish to add load and connection affinity as criteria.

III. Design Description

This section gives a detailed description of NIRN. We take a "layered" approach to the description—we start with the core concepts and basic design and then step-by-step describe additional functionality that satisfies specific goals listed in section II.

NIRN is deployed as an overlay infrastructure. It may be deployed by a CDN company like Akamai, by multiple coop-

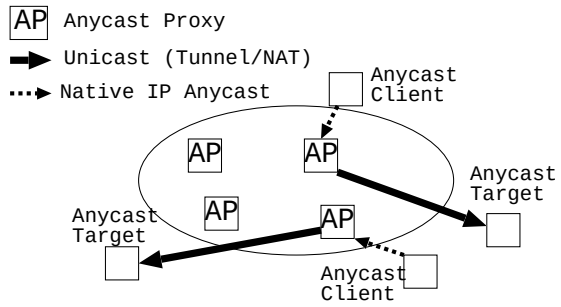


Fig. 1. Proxy Architecture: the client packets reaching the proxies through native IP anycast are tunnelled to the targets

erating ISPs, or even by a single ISP (though the efficacy of proximity discovery would be limited by the ISP's geographic coverage). Multiple distinct NIRN infrastructures may be deployed. In this case, each operates using distinct blocks of IP anycast addresses, and they do not interfere with each other[3]. In the remainder of this document, for simplicity of exposition, we assume a single NIRN infrastructure.

The basic idea of NIRN, illustrated in Figure 1, is very simple. Router-like boxes, hereon referred to as ***anycast proxies*** (AP or simply *proxies*), are deployed at various locations in the Internet, i.e. at POPs (Point of Presence) of different ISPs. These proxies advertise the same block of IP addresses, referred to as the anycast prefix, into the routing fabric (BGP,IGP). As such, they are reachable by native IP anycast—a packet transmitted to the anycast prefix will reach the closest proxy. However, these proxies are not the actual ***anycast target destinations***(AT)[4]. Rather, true to their name, they proxy packets that reach them via native IP anycast to the true target destinations using unicast IP. This proxying can take the form of lightweight tunnels or NAT. NAT allows for backwards compatibility with the protocol stack at target hosts, but makes the processing at the proxy expensive.

This novel combination of native IP anycast with tunnelling to the unicast addresses of the targets allows NIRN to fulfill three critical design goals and drives the rest of the system design. First, it allows for efficient use of the address space as all the IP addresses in the prefix advertised by the proxies can be used by different anycast groups. In fact, NIRN does one better. It identifies an anycast group by the full ***transport address*** (TA), i.e. IP address and TCP/UDP port, thus allowing thousands of anycast groups per IP address. Second, it solves the IP routing scaling problem by allowing so many anycast groups to share a single address prefix and hence, fulfills goal 7. Finally, it relieves targets from the burden of interacting with the routing substrate. They can join an anycast group by registering with a nearby proxy which is discovered using native IP anycast. This fulfills goal 6.

The reader may notice two suspicious claims in the last paragraph. First, we claim to ease deployment by running unicast at the target instead of anycast, and yet the proxies still must run anycast. So, how is this an improvement? The benefit is that the difficult work of deploying IP anycast is borne by the anycast provider once, and amortized across many anycast groups. Second, we claim to improve scaling by allowing thousands of IP anycast groups to share a single IP address prefix. All we've really done, however, is to move the scaling problem from the IP routing domain to the NIRN infrastructure domain. This is quite intentional. As we argue later on, the scaling issues are much easier to deal with in the overlay than in IP routing.

NIRN offers two primitives to the members of an anycast group which involve sending messages to a nearby proxy:

- *join(IP_A:$port_A$,IP_T:$port_T$,options)*: this message instructs the proxy to forward packets addressed to the anycast group identified by the TA IP_A:$port_A$ to the joining node's unicast TA IP_T:$port_T$. The *options* may specify additional information such as the selection criteria (load balance etc.), delivery semantics (scoping etc.), or security parameters needed to authenticate the target host. These are discussed later.
- *leave(IP_A:$port_A$,IP_T:$port_T$,options)*: this message informs the proxy that the target identified by TA IP_T:$port_T$ has left the group IP_A:$port_A$. *options* are the security parameters.

The join-leave messages are transmitted to the anycast address IP_A(which belongs to the anycast prefix) at some well-known port that is dedicated to receiving registration messages. This means that no extra configuration is required for a target to discover a nearby proxy. The service, as described, differs from IPv6's anycast service model, where the join request is sent to a first-hop router, and from ATM's anycast service model, where the join request is sent to an ATM switch.

Note that we don't specify a "create group" primitive. For the purpose of this paper, we assume that the first join essentially results in the creation of the group. In practice, a subscriber to the service would presumably have entered into a contract with the anycast service provider, which would have resulted in the assignment of anycast TAs to that subscriber. The subscriber would also have obtained authentication information using which targets may join the group. While the issues surrounding this sort of group creation are important, they are not central to the NIRN architecture, and we don't discuss them further.

A. The Join Anycast Proxy (JAP)

A target may leave a group either through the leave primitive, or by simply falling silent (for instance, because the target is abruptly shut off or loses its attachment to the Internet). This means that the ***Join AP*** (JAP—the nearby proxy with which the target registers) must monitor the health of its targets, determine when they are no longer available, and treat them as having left the group. The proximity of the JAP to the target makes it ideal for this.

The JAP must also inform zero or more other anycast proxies (APs) of the target(s) that have registered with it. This is because not all APs may be JAPs for a given group (that is, no target joined through them), but **anycast clients** (ACs) may nevertheless send them packets destined for the group. A proxy that receives packets directly from a client is referred to as the ***Ingress AP*** (IAP)[5] for the client. Note that the client-IAP relation is established using native IP anycast. As an IAP, the proxy

[3]Indeed, a single operator could deploy multiple distinct NIRN infrastructures as a way to scale.

[4]**the members of the anycast group; hereon referred to as anycast targets or simply targets**

[5]in figure 1 the proxies in the client-target path are IAPs

must know how to forward packets towards a target (though the IAP may not explicitly know of the target).

One possible way to achieve this would have the JAP spread information about targets associated with it to all proxies. This allows the IAP to tunnel packets directly to clients (as in Figure 1). However, such an approach would hamper NIRN's ability to support a large number of groups. In fact, Figure 1 is conceptual—NIRN's approach for spreading group information is described in the next section and the actual paths taken by packets are shown in Figure 2.

B. Scale by the number of groups

In the previous section, we mentioned the need for a scheme that would allow NIRN to manage group membership information while scaling to a large number of groups. For any given group, we designate a small number of APs (three or four) to maintain a list of JAPs for the group. When acting in this role, we call the AP a ***Rendezvous Anycast Proxy*** (RAP). All APs can act as RAPs (as well as as JAPs and IAPs).

The RAPs associated with any given group are selected with a consistent hash [16] executed over all APs. This suggests that all APs know of all others, and maintain the current up/down status of all others. This is possible, however, because we can assume a relatively small number of global APs ($\sim 20,000$, a number we derive later). We also assume that, like infrastructure routers, APs are stable and rarely crash or are taken out of service. The APs can maintain each other's up/down status through flooding, gossip [31] or a hierarchical structure [27]. The current implementation uses flooding. Such an arrangement ensures that we attain a simple one-hop DHT and hence, limit the latency overhead of routing through the proxy overlay.

When a proxy becomes a JAP for the group (i.e. a target of the group registers with it), it uses consistent hashing to determine all the RAPs for the group and informs them of the join. This allows the RAP to build a table of JAPs for the group. When a RAP becomes unreachable, for each group associated with that RAP, some other AP will become a RAP. The new RAP must now obtain the table of JAPs. Broadly speaking, this is done by having one of the RAPs that is still alive (i.e. the one with the highest node ID) transfer its JAP table to the new RAP.

The concept of the RAP leads to a packet path as shown on the left side of Figure 2. When an IAP receives a packet for an anycast group that it knows nothing about, it hashes the group TA, selects the nearest RAP for the group, and transmits the packet to the RAP (path segment 2). The RAP receives the packet and selects a JAP based on whatever selection criteria is used for the group. For instance, if the criteria is proximity, it selects a JAP close to the IAP. The RAP forwards the packet to the selected JAP (path segment 3), and at the same time informs the IAP of the JAP (the RAP sends a list of JAPs, for failover purposes).

The use of RAPs unfortunately introduces another overlay hop in the path from client to target. We mitigate this cost however by having the IAP cache information about JAPs. Once the IAP has cached this information, subsequent packets (not only of this connection, but of subsequent connections too) are transmitted directly to the JAP. This is shown in the right-hand side of Figure 2. The time-to-live on this cache entry can be quite large. This is because the cache entry can be actively invalidated in one of two ways. First, if the target leaves the JAP, the JAP can inform the IAP of this when a subsequent packet arrives. Second, if the JAP disappears altogether, inter-AP monitoring will inform all APs of this event. In both cases, the IAP(s) will remove the cached entries, failover to other JAPs it knows of, or failing this, go back to the RAP. Because of this cache invalidation approach, the IAP does not need to go back to the RAP very often.

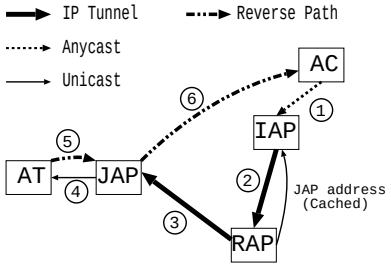
Note that in figure 2 the JAP is responsible for transmitting packets to and receiving packets from its targets. The reason for this is not obvious. The IAP could well have cached the target information (instead of the JAP), allowing for the packet to take the client-IAP-target path in the forward direction, and target-client path in the reverse direction. This latter path, however, is not possible with legacy clients, which expect to see return packets coming from the same address and port to which they sent packets. In general, targets cannot source packets from anycast addresses, because nearby routers would detect this as spoofed source addresses and drop the packets. So, at least one AP must be inserted into the target-client path. Furthermore, if NAT is being used to forward packets to the target, then the AP with the NAT state should be the AP that handles the return packets.

This might argue for traversing the IAP in the reverse direction too, since by necessity it must be traversed in the forward direction. The argument in favor of using the JAP however, boils down to the following two points. First, it is highly convenient to keep all target state in one proxy rather than two or more. Since the JAP in any event must monitor target health, it makes sense to put all target state in the JAP. Second, the JAP is close to the target, so the cost of traversing the JAP in terms of path length is minimal (Section IV-C). Also, by seeing packets pass in both directions, the JAP is better able to monitor the health of the target. For the most part, when a packet passes from client to target, the JAP may expect to soon see a packet in the reverse direction. Rather than force the JAP to continuously ping each target, the lack of a return packet can be used to trigger pings.

The use of proxies implies that the NIRN path (AC⇒IAP⇒JAP⇒AT) might be longer than the direct path (AC⇒AT)[6]. If the client reaches the target over the WAN, the proximity of the client to the IAP and of the target to the JAP should ensure that NIRN imposes minimal stretch and hence fulfills goal 3. This has been substantiated by simulating the stretch imposed by NIRN across a tier-1 topology map of the Internet. The cases where this does not hold are: 1) if the client and target are on the same LAN, 2) if the client and target are in the same private network. These two cases are briefly discussed in Section VII.

The introduction of the RAP to allow scaling by the number of groups is somewhat equivalent to the extra round-trip imposed by application-level anycast schemes, for instance in the form of the DNS lookup or the HTTP redirect. This is one aspect of NIRN that falls short of native IP anycast, which has no such extra hop. Having said that, it would be possible for

[6]the NIRN path may be shorter as inter-domain routing is not optimal[20]



No.	Source	Dest	Comment
1	AC:p	AA:g	Native IP Anycast
2	IAP:AC:p	RAP:AA:g	IP-IP tunnel
3	RAP:AC:p	JAP:AA:g	IP-IP tunnel
4	JAP:q	AT:r	Unicast IP JAP:q = NAT(AC:p)
5	AT:r	JAP:q	Unicast IP
6	AA:g	AC:p	Unicast IP AC:p = NAT^{-1}(JAP:q)
7	IAP:AC:p	JAP:AA:g	IP-IP tunnel

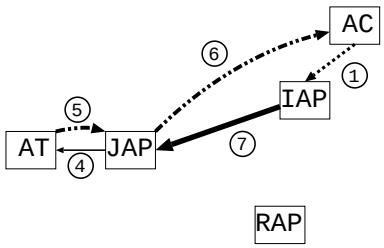


Fig. 2. Initial (leftmost) and subsequent (middle) packet path. The table shows the various packet headers. Symbols in block letters represent IP addresses, small letters represent ports. AA(Anycast Address) is one address in the address block being advertised by NIRN, AA:g is the transport address assigned to the group the target belongs to, while AT:r is the transport address at which the target wants to accept packets. Here, the target joined the group by invoking join(AA:g,AT:r,NULL)

a small number of groups with minimal target churn to operate without RAPS—that is, to spread JAP information among all APs. This might be appropriate, for instance, for a CDN or for 6to4 gateways. By-and-large, however, we can expect most groups to operate with RAPs as described here, and in the remainder of the design section, we assume that that is the case.

C. Scale by group size and dynamics

If the only selection criteria used by a RAP to select a JAP were proximity to the client, then the RAP could ignore the number of targets reachable at each JAP. In order to load balance across targets, however, RAPs must know roughly how many targets are at each JAP. In this way, RAPs can select JAPs in a load balanced way, and each JAP can subsequently select targets in a load balanced way. Unfortunately, requiring that RAPs maintain counts of targets at JAPs increases the load on RAPs. This could be a problem for very large groups, or for groups with a lot of churn.

We mitigate this problem by allowing the JAP to give the RAP an approximate number of targets, for example within 25% or 50% of the exact number. For instance, if 25% error is allowed, then a JAP that reported 100 targets at one time would not need to report again until the number of targets exceeded 125 or fell below 75. This approach allows us to trade-off the granularity of load-balancing for scalability with group size and dynamics. Indeed, this trade-off can be made dynamically and on a per-group basis. A RAP that is lightly loaded, for instance, could indicate to the JAP that 100% accuracy reporting is allowed (i.e. in its acknowledgement messages). As the RAP load goes up, it would request less accuracy, thus reducing its load. The combination of the two-tiered approach with inaccurate information in a system with 2 groups is illustrated in Figure 3 (the figure assumes that there is just one RAP for each group). Section IV-B presents simulations which show the benefits of this approach in the case of a large, dynamic group.

In any event, the number of targets is not the only measure of load. Individual targets may be more-or-less loaded due to differing loads placed by different clients. Ultimately, the JAP may simply need to send a message to the RAPs whenever its set of targets are overloaded for whatever reason.

D. Scale by number of proxies

Up to this point, we have laid out the basic architecture of NIRN, and justified that architecture in terms of certain performance goals. Now we can look specifically at NIRN deployment issues. A central question is, how many proxies may we

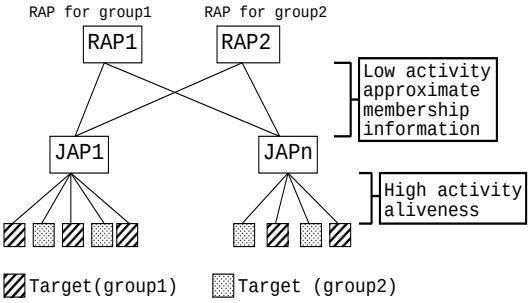


Fig. 3. 2-tier membership management: the JAPs keep the aliveness status for the associated targets; the RAP for a group tracks the JAPs and an approximate number of targets associated with each JAP

reasonably expect in a mature NIRN deployment, and can we scale to that many proxies?

A key observation to make here is that the scaling characteristics of NIRN are fundamentally different from the scaling characteristics of IP routing. While the traffic capacity of the Internet can be increased by adding routers, the scalability of IP routing per se is not improved by adding routers. All routers must contain the appropriate routing tables. For instance, all Tier1 routers must contain the complete BGP routing table no matter how many Tier1 routers there are. For the most part, IP routing is scaled by adding hierarchy, not adding routers.

With NIRN, on the other hand, scaling does improve by adding proxies. With each additional proxy, there are lower ratios of target-to-JAP and group-to-RAP. Growth in the number of groups and targets can be absorbed by adding proxies. However, an increase in the number of proxies presents its own scaling challenge. Among other things, every proxy is expected to know the up/down status of every other proxy.

There is a simple divide-and-conquer approach we may take if the number of proxies grows too large. Split the IP anycast prefix advertised by the proxies in two (say, P1 and P2), and have half the proxies advertise P1, and the other half advertise P2. These two groups of proxies now don't have to know anything about each other—they are completely distinct groups. In other words, just as different organizations (CDNs, ISPs) can operate distinct NIRN infrastructures by virtue of using different IP anycast prefixes, a single organization can scale its NIRN infrastructure indefinitely by defining distinct IP anycast prefixes and partitioning groups of proxies accordingly.

Thus, a given anycast service provider may scale its NIRN infrastructure as follows. It starts with one anycast prefix, and deploys proxies in enough geographically diverse POPs to achieve good proximity. As more anycast groups are created, or as

existing anycast groups grow, the provider expands into more POPs, or adds additional proxies at existing POPs. With continued growth, the provider adds more proxies, but it also obtains a new IP anycast (or splits the one it has), and splits its set of proxies into two distinct groups. Because the IP routing infrastructure sees one address prefix per proxy group, and because a proxy group can consist of thousands of proxies and tens of thousands of anycast groups, the provider could continue adding proxies and splitting proxy groups virtually indefinitely.

The need for geographic diversity as well as some minimum number of proxies per POP for robustness (described later in the section) suggests that there is some minimum number of proxies a given proxy group should have. That number may be roughly calculated as follows. There are about 200 tier-1 and tier-2 ISPs [37]. An analysis of the ISP topologies mapped out in [19] shows that such ISPs have $\sim$25 POPs on average. Assuming that we'd like to place proxies in all of these POPs, this leads to 5000 POPs. Assuming 3-4 proxies per POP, for reliability, this implies a conservative total of roughly 20,000 proxies before we can split the infrastructure.

While 20,000 proxies is not an outrageous number, it is large enough that we should pay attention to it. One concern not yet addressed is the effect of the number of proxies on IP routing dynamics. In particular, BGP reacts to route dynamics (flapping) of a single prefix by "holding down" that prefix—ignoring any advertisements about the prefix for a period of roughly one hour [26]. A naive proxy deployment where each proxy advertises the anycast prefix directly into BGP would imply that a proxy failure necessitates a BGP withdrawal for the prefix (from the site where the proxy is located) which could lead to dampening. While the proxy stability ensures that such events do not occur often, even the occasional prefix instability and the consequent service disruptions that a large proxy deployment would entail are not acceptable.

As a result, we cannot have the proxies advertising the anycast prefix directly into BGP. Rather the deployment model involves more than one proxy being placed inside every POP where the proxies are deployed. Such an arrangement is referred to as an anycast cluster[7] and is based on the model used by the anycasted f root-server[25] (shown in figure 4). The approach involves connecting one or more routers and more than one proxy to a common subnet. All the proxies in the cluster advertise the anycast prefix into IGP. So, the routers receive equal cost routes for the anycast prefix. These routers use flow-hashing to ensure that packets of the same flow are routed to the same proxy i.e. client packets are delivered to the IAP for the corresponding flow. This deployment model ensures that an individual proxy failure does not result in a BGP withdrawal.

E. Proximity

The introduction of the proxies into the IP path negates the natural ability of native IP anycast to find the nearest target. Therefore, we require explicit mechanisms in NIRN to regain this capability.

As mentioned before, native IP anycast sets the client-IAP and target-JAP path segments. The RAP, on the other hand, selects the JAP, and therefore sets the IAP-JAP path segment (on forward packets) and the JAP-client path segment (on return packets). To ensure the proximity of the target to the client, the RAP must choose a JAP close to the IAP and hence, every AP must know the distance (in terms of latency) between every pair of APs. This could be accomplished using a proximity addressing scheme like GNP [29] or Vivaldi [28].

We believe, however, that a simple, brute-force approach, whereby every AP occasionally pings every other AP and advertises the minimum measured round trip time (RTT) to all other APs, will suffice. This is because, with the cluster deployment approach, RAPs in fact only need to know the distance between each pair of clusters. Using the previously derived number of 5000 clusters (each in a different POP), this translates into 25M cluster pairs. Assuming that we don't try to make the measurements real-time, distributing 25M measurement values appears straightforward.

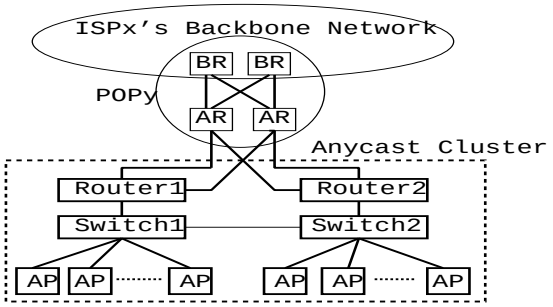


Fig. 4. Anycast cluster: the proxies(APs) advertise the prefix to the routers through IGP and the routers maintain a E-BGP relation with the POP's access routers(AR). BR stands for backbone routers.

F. Robustness and fast failover

The introduction of proxies between client and target might have a negative impact on the robustness as compared to native IP anycast. On the other hand, RON[20] has shown how an overlay structure can be used to improve the resiliency of communication between any two overlay members. Extending the same thought, NIRN, by ensuring the robustness of packet traversal through the proxy overlay, can improve the resiliency of communication service between clients and group members. We believe that given the stable nature of the proxies, their deployment in well connected parts of the Internet (tier-1 and tier-2 ISPs) and the engineering that would go into their set-up, NIRN should be able to match, if not better, the robustness offered by native IP anycast.

A related requirement is that of fast fail-over. "E2E" native IP anycast has to achieve failover when a group member crashes so that clients that were earlier accessing this member are served by some other group member. And given the way native IP anycast works, this failover is tied to IP routing convergence. Specifically, in case of a globally distributed group, the failover is tied to BGP convergence, which in some cases can extend to a few minutes[20]. Since NIRN uses native IP anycast to reach the proxies, it is subject to the same issues. The process of overcoming the failure of a proxy is termed as **proxy failover**. In addition, the proxies must themselves be able to fail over from one target to another which is termed as

[7]hereon referred to as proxy cluster or simply, cluster

Segment	Failure of	Failover through	Section
AC⇒IAP	IAP	IGP, onto a proxy within the same cluster	III-F.2
IAP⇒JAP	JAP	proxy health monitoring system	III-F.2
JAP⇒AT	AT	pings between target and JAP, passive monitoring by JAP	III-A,III-B
AT⇒JAP	JAP	pings routed to a different proxy who becomes JAP	III-F.2
JAP⇒AC	AC	client fails	-

TABLE I

FAILOVER ALONG THE NIRN FORWARD PATH (AC⇒IAP⇒JAP⇒AT) AND REVERSE PATH (AT⇒JAP⇒AC)

target failover. Thus the failover problem seems worse with NIRN than with native IP anycast.

1) Target failover: As discussed in Sections III-A and III-B, the JAP is responsible for monitoring the aliveness of its targets. It does this through pinging and tracking data packets to and from the target. The JAP is also responsible for directing IAPs to delete their cache entries when enough targets have failed.

2) Proxy failover: There is still the question of clients failing over onto a different proxy when their IAP crashes and targets failing over when the corresponding JAP crashes. And there are two levels at which this must be achieved: at the routing level and at the overlay level.

At the routing level, the system must be engineered such that when a proxy fails, clients which were using this proxy as an IAP are rerouted to some other proxy quickly. NIRN's deployment of proxies in a cluster means that this failover is across proxies within the same cluster. Also, since the proxies advertise the prefix into IGP, NIRN relies on IGP for convergence after a proxy failure and hence can achieve faster failover. Typically this is of the order of a few seconds and can be reduced to sub-second times[1].

At the overlay level, to monitor the health of proxies, we use a 2-tier health monitoring system. At the first tier, the proxies within the same proxy cluster are responsible for monitoring each other. At the next level, each proxy in a cluster monitors the health of a small number of other clusters. When either an individual proxy or an entire cluster fails, it is detected quickly and communicated to all remaining proxies.

Section III-B had described JAP and IAP behavior when a RAP goes down, as well as IAP behavior when a JAP goes down. The only thing left to discuss is target behavior when a JAP goes down. In this case, native IP anycast routing will cause ping packets from the target to reach another JAP, which will ask the target to re-register. Table I sums up the way NIRN achieves failover across various segments of the client-target path.

G. Target selection criteria

As described earlier, the RAP may select the JAP based on a number of criteria, including proximity, load balancing, and connection affinity[8]. The JAP subsequently selects a target. It is this selection process, divorced from IP routing, that allows NIRN to offer richer target selection criteria. As such, this aspect of NIRN deserves more discussion.

[8]Connection affinity—all packets from a given connection or flow are delivered to the same target.

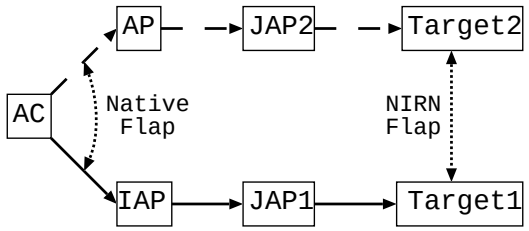


Fig. 5. Lack of native IP anycast affinity can cause flaps in the NIRN model

How NIRN achieves load balance and proximity has already been discussed. Connection affinity is discussed later in this section. We wish to point out here that these three important selection criteria are in fact at odds with each other. If both load balance and proximity are important criteria, and the JAP nearest to the IAP is heavily loaded, then one of the other criteria must be compromised. Likewise, a target selected earlier in a connection may have since become heavily loaded, but connection affinity would dictate that packets continue to be delivered to the target. This basic set of trade-offs applies to application-level anycast as well.

As mentioned earlier, NIRN allows a target in a given anycast group to be a client of that group. Packets sent by the target to the group address would be forwarded to the nearest group target other than the sender. Note that this is not possible with native IP anycast and it allows NIRN to support new P2P applications, for instance, those described in section VI.

Proxies could potentially base their target selection on still other criteria. These selection criteria can be expressed by overloading the transport address, i.e. a group can have separate TAs for each type of scoping. For instance, an anycast packet could be administratively scoped. That is, it could in theory indicate that the target should be in the same site, belong to the same DNS domain, or have the same IP address prefix (or be from different sites, DNS domains, or IP prefixes). This would necessitate the RAP having extra knowledge about the targets than in the current implementation—for example, for selection based on IP address (unicast) prefix, the RAP would need to know the prefix of the targets associated with each JAP. While how this would be configured and operated is a good topic for further study, the selection functionality of the RAP allows for the possibility of many such features.

Another form of selection would be to pick a random target rather than the nearest target - the RAP would pick a random JAP who would then pick a random target. Random selection among a group can be useful for various purposes such as spreading gossip [24] or selecting partners in a multicast content distribution [23]. Indeed, in the NIRN architecture, there is no reason an anycast packet cannot be replicated by the RAP and delivered to a small number of multiple targets. The salient point here is that, once IP anycast functionality is divorced from IP routing, any number of new delivery semantics are possible if the benefits justify the cost and complexity.

1) Connection affinity: Lack of connection affinity in native IP anycast has long been considered one of its primary weak points. Indeed, this concern has caused the IPv6 community to

mandate that an anycasted address may not be used as a source address (lest the reply from a transmitted packet go to another host). This issue spills over into NIRN. Specifically, the issue is how to maintain affinity when native IP anycast causes a different IAP to be selected during a given client connection. If the same IAP is always used, then packets will be sent to the same JAP that was initially cached by the IAP. However, a change in the IAP could lead to a change in the target the packets are delivered to, as shown by Figure 5. Application-layer anycast doesn't have this problem, because it always makes its target selection decision at connection start time, and subsequently uses unicast.

A simple solution would be to have RAPs select JAPs based on the identity of the client, such as the hash of its IP address. This way, even if IP routing caused packets from a given client to select a different IAP, they would be routed to the same JAP and hence the same target. Unfortunately, this approach completely sacrifices proximity and load balance. Broadly, another approach would be to modify the host application by making it anycast aware, and redirect the host to the unicast address of a selected target (either NIRN or the target itself could do this redirect). There are some security issues here—the redirect must be hard to spoof—but these are surmountable.

We can also imagine complex schemes whereby JAPs and IAPs coordinate to insure affinity. However, a fundamental question that still has not been answered is, how good or bad is the affinity offered by native IP anycast? It might be the case that the affinity offered by native IP anycast is very good; i.e. the probability that a connection breaks due to a routing flap is very small as compared to the probability of the connection breaking due to other factors. This would imply that we do not need the complex mechanisms stated above. In this regard, we did some measurements to find out the affinity offered by native IP anycast. Our results, while preliminary, suggest that native IP anycast affinity is pretty good, and NIRN need not do anything extra to provide connection affinity. Details of these measurements are presented in section IV-A

IV. Evaluation

This section evaluates various aspects of the NIRN architecture. Section IV-A uses measurement results made using the Planetlab[17] testbed and the anycasted DNS root servers to argue for the sufficiency of the affinity offered by native IP anycast and hence, NIRN. Sections IV-B and IV-C present simulation results which show the scalability (by group characteristics) and the efficiency of the NIRN deployment. Section IV-D looks at the quality of proximity offered by native IP anycast, again using measurements. Finally, section IV-E discusses our NIRN implementation and presents relevant micro-benchmarks.

A. Connection Affinity measurements

As mentioned earlier, it is important to determine the affinity offered by native IP anycast in order to understand the need for mechanisms to ensure affinity in NIRN. This section presents the results of our measurement study aimed to do so. The goal of the study was to determine how often IP routing selected different locations when sending packets to a native IP anycast address. We used the anycasted root-servers and the AS-112 servers as the anycast destinations. The number of locations for each of these anycasted destinations is given in table II. For clients, we used 129 Planetlab nodes, belonging 112 sites.

Anycasted Server	Number of locations
C-root	4
F-root	28
I-root	17
J-root	13
K-root	11
M-root	3
AS112	20

TABLE II
ANYCASTED SERVERS AND LOCATIONS

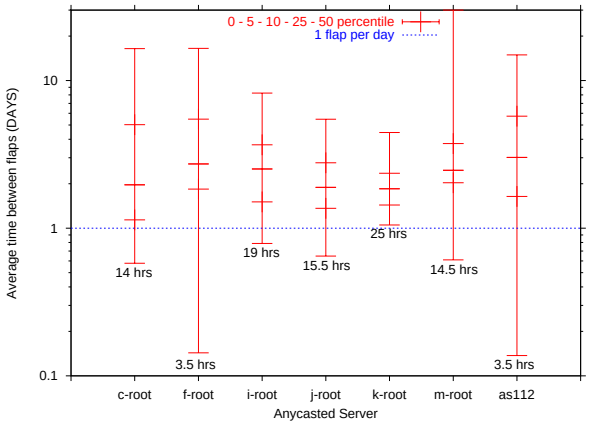


Fig. 6. Percentiles for the average time between flaps for all the anycasted destinations

For each anycast destination, the clients probed the associated anycast address every 10 seconds to determine the location they are routed too. The servers at different locations have been configured by their operators to respond to a TXT type DNS query with their location[18] and hence, the probes were DNS queries generated using *dig*. This data was collected for a period of 30 continuous days in Dec'04-Jan'05.

The probing of the anycasted destinations reveals changes in routing or 'flaps' which cause packets to be delivered to different locations of an anycasted server. So, two subsequent packets from a given Planetlab node switching from the San Jose f root-server to the Palo Alto f root-server[9] would be counted as one flap. Figure IV-A plots various percentiles for the average time between flaps. The figure shows that the anycasted services are very stable as viewed from almost all locations. For example, more than 95% of the nodes observed less than a flap per day for all the anycasted destinations. Similarly, ∼48% of the nodes never observed a flap when probing the f-root during the entire 30 day period.

Also, the few nodes which observed frequent flaps (i.e. an average inter-flap duration of less than a day) had their average skewed by tiny bursts of instability in between large periods of stability. For example, the Planetlab node which experienced most flaps (208) over the month when probing j-root was in Leixip, Ireland. Of these, 180 flaps occurred in a 3-hour period. We conjecture that such phenomena can be attributed

[9]San Jose and Palo Alto are two locations of the f root-server

to ephemeral issues specific to the sites to which these nodes belong. While a more rigorous analysis of the collected data and correlation with BGP-updates for the prefixes representing these anycasted destinations would be needed for determining the causes and patterns amongst these flaps, the overall figures do paint an encouraging picture. These measurements reveal that the probability that a two minute connection breaks due to a flap is about 1 in 4500 and the probability that an hour long connection breaks is about 1 in 150. Note that it is the short connections that, in order to avoid the overhead of anycast to unicast redirect, need to rely on anycast affinity. Long connections can incur the overhead of a redirect and hence, would use anycast for discovery and unicast for the actual communication. All this points to the negligible quantitative impact of the use of anycast for connection oriented services; anycast certainly does not make things qualitatively worse.

We admit that the limited number of vantage points (129) and the number of locations of the anycast destinations makes our study preliminary. Also, the operators of j-root, based on their observations, have come to the opposite conclusion regarding the ability of native IP anycast to support stateful connections[33]. While their results are being debated by many in the operational community[36], we are trying to acquire the relevant data-sets so as to find the reason for the flapping observed by them (something which the authors of the j-root study have not analyzed).

B. Scalability by group size and dynamics

In this experiment, we evaluate NIRN's ability to handle large and dynamic groups (as described in III-C). We simulate the load imposed by a large group with high churn on the proxy infrastructure. The dynamics of the simulated group - the arrival rate of group members and the session duration cumulative distribution function - resemble the dynamics of the largest event observed in a study of large-scale streaming applications[21]. Simulation of just one such group is sufficient as the load imposed varies linearly with the number of such groups supported.

The NIRN infrastructure simulated varies the number of proxies and maximum group size. We simulate four RAPs per group. We want to measure the number of messages required to keep the 2-tier membership hierarchy updated in face of the group dynamics. This is the number of messages from the JAPs of the group to the 4 RAPs and is referred to as 'system wide messages'.

Figure 7 plots the system wide messages produced with a proxy deployment of size 1000 and the group size bounded by 90000. The topmost curve in the figure shows how the group size varies with the time. A flash crowd, at a rate of $\sim$100 members/second, leads to a sudden rise in the group size in the first 10 minutes. The other curves plot the number of messages produced in the corresponding minute (as plotted along the X-axis) for varying degrees of inaccuracy. The degree of inaccuracy, as explained in section III-C, implies that a JAP only informs a RAP of a change in the number of members associated with it if the change is more than a certain percentage of the last value sent.

The inaccuracy of information offers a small benefit in the nascent stages of the group (the first minute). This is because no matter what inaccuracy percentage we use, the JAP must inform the RAP of the first group member that contacts it. In the next couple of minutes, as the group increases in size and more members join their corresponding JAPs, the inaccuracy causes the traffic towards the 4 RAPs to drop rapidly (see the embedded graph in figure 7). Overall, the average number of messages over the duration of the entire event reduces from 2300 per min. with the naive approach to 117 per min. with 50% inaccuracy.

Figure 8 plots the average system wide messages (per second) versus the percentage of inaccuracy for varying number of proxies and varying maximum group size. Each plotted point is obtained by averaging across 20 runs. All curves tend to knee around an inaccuracy mark of $50\%-60\%$. The closeness of the curves for different sized groups (given a fixed number of proxies) points to the scalability of the system by the group size even in the face of high churn.

More interesting is the variation of the load on the RAPs with the number of proxies. As the number of proxies increase, the number of JAPs increase; an offshoot of the assumption that the group members are evenly distributed across the proxy infrastructure. For a given group size, each JAP is associated with lesser number of group members. and hence, there is lesser benefit due to the inaccuracy approach. This shows up as the increase in the average number of messages directed towards the RAPs with the number of proxies.

The figure shows that such an extreme group in a 100 proxy deployment with 100% inaccuracy would require an average of $\sim$0.18 messages/second. As a contrast the same setup in a 10000 proxy deployment would necessitate an average of $\sim$7.25 messages/second. The low message overhead substantiates the NIRN scalability claim. Note that a larger number of proxies implies that each proxy is a RAP for a smaller number of groups. The number of targets associated with each proxy (as a JAP) reduces too. Thus, increasing the number of proxies would indeed reduce the overall load on the individual proxies.

C. Stretch

NIRN causes packets to follow a longer path (client $\Rightarrow$ IAP $\Rightarrow$ JAP $\Rightarrow$ target). We have argued that the combination of native IP anycast and proxy-to-proxy latency measurements minimizes the effect of this longer path. This section simulates the stretch introduced by NIRN along the end-to-end path.

For the simulation, we use a subset of the actual tier-1 topology of the Internet, as mapped out by Spring et. al. in [19]. This subset consists of 22 ISPs, 687 POPs, and 2825 inter-POP links (details in [42]). The use of only a subset of the tier-1 topology, which was necessitated by limitations in the SSFNET BGP simulator, can be justified on two grounds. First, a large proportion of traffic between a randomly chosen client-target pair on the Internet would pass through at least one tier-1 ISP. Second, such a simulation gives us an approximate idea about the overhead that a NIRN deployment restricted to tier-1 ISPs would entail.

The topology was annotated with the actual distance between POPs (in Kms) based on their geographical locations. We then used SSFNET[40] to simulate BGP route convergence. This allowed us to construct forwarding tables at each of the POPs and hence, determine the forwarding path between any two POPs.

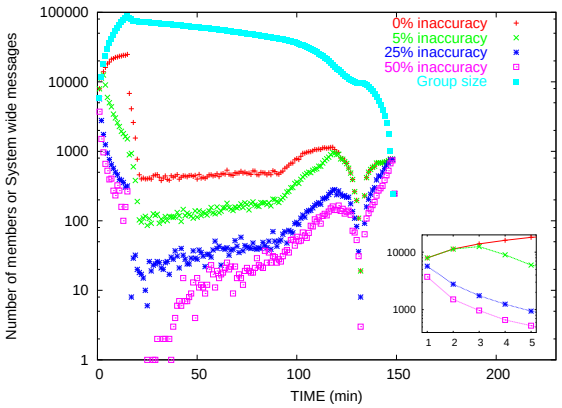


Fig. 7. System wide messages from the all the JAPs to the 4 RAPs during the event for varying degrees of inaccuracy

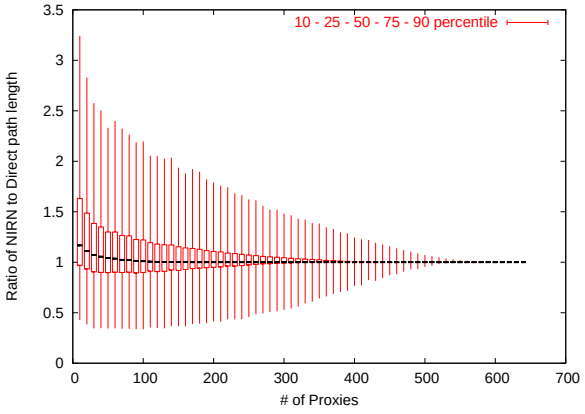


Fig. 9. Percentiles for the stretch with varying number of proxies

The simulated NIRN deployment involves placing a variable number of proxies at random POPs, one proxy per POP. These POPs are referred to as the *proxy POPs*. For every client-target pair to be simulated, we choose a POP through which the client's packets enter the topology (the *client POP*) and a POP through which the target's packets enter the topology (the *target POP*). The forwarding paths between the client and the target through these POPs represents the *direct path*. The IAP is assumed to be in the *proxy POP* closest to the client POP—this is the *IAP POP*. Similarly, the JAP is in the *proxy POP* closest to the target POP—this is the *JAP POP*. The *NIRN path* comprises of the following three segments: from the *client POP* to the *IAP POP*, from the *IAP POP* to the *JAP POP* and from the *JAP POP* to the *target POP*.

Figure 9 plots the percentiles for the stretch with varying number of proxies. For a given number of proxies, we simulated 100000 runs. Each run comprised of simulating a client-target pair and finding the *direct* and the *NIRN path* length (in kms). Note that the well-documented non-optimal nature of inter-domain routing[20] is reflected in the cases where the NIRN path turns out to be shorter than the direct path. The figure shows that with a deployment of just 100 proxies (a mature deployment might encompass 50 times more POPs), the median stretch is 1.01 with the 90^{th} percentile being 2.2. Hence, even with a small size deployment, NIRN performs well with regards to the direct path.

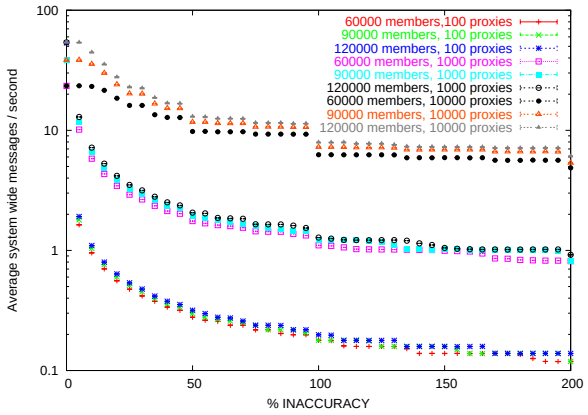


Fig. 8. Average system wide messages (per second) versus the percentage of inaccuracy with varying number of proxies and varying maximum group size.

D. Proximity measurements

NIRN's argument for minimal stretch relies in part on the assumption that the client is close to the IAP and the server is close to the JAP. The assumption is justified by the fact that these relations are discovered using native IP anycast and hence, the distances are small in terms of metrics used by inter-domain routing. However, this does not necessarily imply that the distances are small in terms of latency. An important concern is the ability of native IP anycast to find proxies close in terms of latency.

This experiment involves measuring the latency to an anycast destination (referred to as anycast latency) and the unicast latencies to all the individual locations of the same anycast destination from a given client. Ideally, we want the anycast latency to be the same as the minimum unicast latency. We used King[38] to estimate the aforementioned latencies. As clients, we used one host (rather, one IP address) belonging to each routable prefix in BGP. We were able to measure the above data for 26,553 hosts[10].

Figure 10 shows the CDF for the anycast to minimum unicast latency ratio for the j root-server deployment. With a median of 1.87 and 90^{th} percentile of 8.77, it seems that native IP anycast does not do a great job of selecting good locations, at least not for this particular deployment. This might be due to the way the root-servers have been deployed - all 13 anycasted servers for j-root are placed in POPs of different ISPs. A possible problem with this approach is illustrated in figure 11.

The figure shows 2 ISP networks- I1 and I2, each having a POP in New York and in Berkeley. It also shows a native IP anycast deployment (AS number J) with two servers - one hosted at the New York POP of I2 (I2-NY) and the other at the Berkeley POP of I1 (I1-B). The figure has these POPs highlighted. The anycast servers have an EBGP relation with the routers of the hosting POP; hence, the anycast prefix is advertised with J as the origin AS. Now, if a client (C) in the New York area sends packets to the anycast address and these reach POP I1-NY, they will be routed to the server hosted at I1-B. This is because the routers in I1-NY would prefer the 1 AS-hop path ([J]) through I1-B to the anycasted server over the 2 AS-hop path ([I2,J])

[10]see section II.B in [38] for limitations on measurable clients

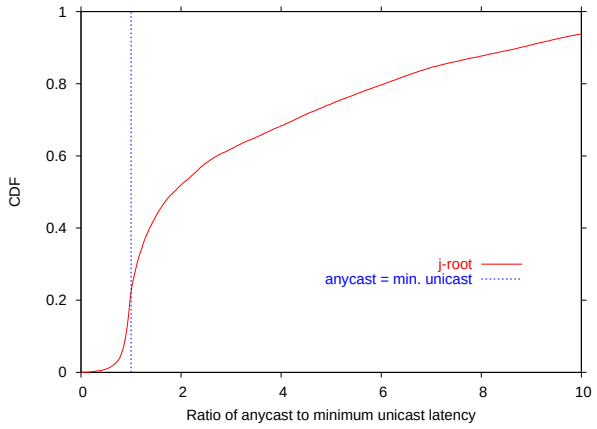


Fig. 10. CDF for the ratio of the anycast latency to the minimum unicast latency.

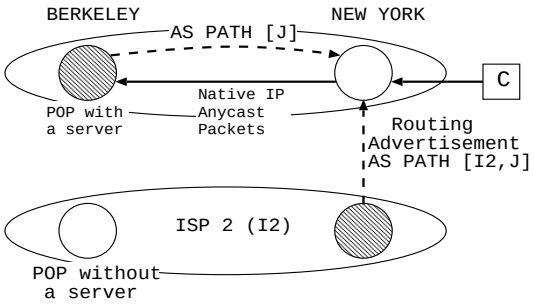


Fig. 11. Native IP anycast inefficiency - packets from the client C in new york destined to the native IP anycast address are routed to the anycast server in berkeley, even though there is a server in new york

through I2-NY. Note that the anycasted server hosted at I1-B represents a customer of I1 and so, it would very uncommon for I1 to steer these packets towards I2-NY due to local policies (local preference values); rather the AS path length would decide the path the packets take.

Although negative, the importance of the result cannot be overemphasized. It brings out the fact that a naive proxy deployment might not achieve low-latency client-IAP and JAP-target paths. Also, an unverified implication of the above analysis is that for good performance, an ISP that is part of the deployment[11] should be sufficiently covered, i.e., there should be clusters at a decent number of POPs of the ISP. For example, deployment of the two servers in the figure at both of the POPs of I1 (I1-NY and I1-B) or I2 (I2-NY and I2-B) would avoid the problem of long paths. We believe that such an approach would ensure that the client-IAP and the target-JAP segments are latency-wise small - something that can only be substantiated when we get the NIRN deployment going. In this regards, we are looking at several possible deployment opportunities and have acquired an address block (a /22) and an AS number from ARIN for this purpose.

E. Implementation micro-benchmarks

We have implemented and tested the basic NIRN system in the laboratory. With the system geared towards router-like boxes, the current implementation of NIRN proxies comprises of 2 components:

[11] the ISP has at least one POP hosting a proxy cluster

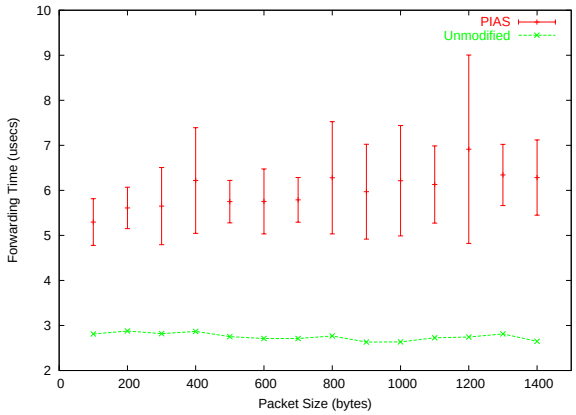


Fig. 12. Forwarding time for the NIRN implementation and the unmodified kernel

1. A user-space component responsible for the overlay management tasks, such as handling proxy failures, target join/leaves, health monitoring etc.
2. A linux kernel-space component responsible for the actual forwarding of packets through the use of Netfilter hooks[22]. This involves tunnelling of the packets when sending them between 2 proxy nodes, and using a NAT when handling packets to/from a target.

To characterize the overhead of forwarding the packets from the IAP to the JAP, we measured the forwarding time. This includes the time to look-up the cached JAP address and wrap the packet in an IP header as part of tunnelling it to the JAP. Note that this is not the time it takes the packet to move from the incoming wire to the outgoing wire. The experiment was performed on Pentium IV 1700 MHz machines running Linux 2.4, while the time was measured using the Pentium timestamp counter (TSC).

Figure 12 plots the forwarding time for the NIRN scheme and the normal kernel. The doubling up of the forwarding time in the NIRN scheme implies that the forwarding capacity of the proxy node is more than half the capacity of the unmodified kernel (as this is not the wire-to-wire time). Of course, with an efficient implementation of the system data structures, this overhead could be reduced even more.

V. Related work

Table III summarizes the pros and cons of NIRN, application level anycast, and other related approaches described below.

Partridge et. al. [2] originally proposed the IPv4 anycast service. Later, IPv6 incorporated anycast into its addressing architecture[6]. It allowed for scoped anycast addresses for groups confined to a topological region which does not burden the global routing system. However, a globally spread group still poses scalability problems. Besides, IPv6 anycast also inherits all the other limitations of IPv4 anycast. Despite the shortcomings, there has been work detailing the relevance of anycast as a tool for service discovery and other applications, both for IPv4[13] and for IPv6[32].

Katabi and Wroclawski[10] proposed an architecture that allows IP anycast to scale by the number of groups. Their approach is based on the observation that services have a skewed

Criterion (related to goal number)	IPv4	IPv6	IP + GIA	App. Level	*i*3	NIRN
Router Modification(1)	No	No	**Yes**	No	No	No
Client Modification(1)	No	No	No	No	**Yes**	No
Scalability by group size(2)	Very Good	Very Good	Very Good	**Poor**	**Poor/Good**[12]	Good
Stretch(3)	No	No	Little/No	No	Little	Little
Robustness(4)	No Issues	No Issues	No Issue	Mixed	Mixed	Mixed[13]
Failover(5)	Fast[14]	Fast[14]	Fast[14]	Fast	Fast	Fast
Target Deployment(6)	**Difficult**	**Difficult**	**Difficult**	Easy	Easy	Easy
Scalability by no. of groups(7)	**No**	**No**	Yes	Yes	Yes	Yes
Scalability by group dynamics(8)	**Poor**	**Poor**	**Poor**	**Poor**	**Poor/Good**[12]	Good
Cost of Proximity(9)	None	None	Small	**Large**	**Large**	Small
Low-level access	Yes	Yes	Yes	**No**	Yes	Yes

TABLE III
THE ANYCAST DESIGN SPACE

popularity distribution. Hence, making sure that the unpopular groups do not impose any load on the routing infrastructure addresses the scalability issue. However, the need to change routers puts a severe dent on the practical appeal of the approach. Besides, being a router-based approach, it suffers from most other limitations of IPv4 anycast.

Because of the limitations of these approaches, anycast today is typically implemented at the application layer. This offers what is essentially anycast service discovery—DNS-based approaches use DNS redirection while URL-rewriting approaches dynamically rewrite the URL links as part of redirecting a client to the appropriate server. These application layer approaches are easier to deploy (no router modifications), provide fine grained control over target server load, and naturally maintain connection affinity.

Related proposals in the academic community include [14][15]. The idea behind these is to identify the group using an application level name which, at the beginning of the communication, is mapped to the unicast address of a group member. Like the techniques mentioned in the last paragraph, these approaches rely on only unicast support from the underlying IP layer and hence, circumvent all the IP anycast limitations. The challenge here is to collect the relevant selection metrics about the group members in an efficient and robust fashion.

Another element in this design space is anycast built on top of the indirection architecture offered by i3[11]. i3 uses identifiers as a layer of indirection which generically gives the receiver tremendous control over how it may (or may not) be reached by senders. One of the services i3 can provide is anycast. Therefore, we need to compare NIRN's anycast service to i3's anycast service. But there is a broader comparison that can be made with i3: they both use indirection. Therefore, we also briefly compare NIRN with i3 in its capacity as a generalized indirection service.

Regarding anycast service, there are two main advantages of NIRN over i3. First, NIRN requires no changes in the protocol stack, whereas i3 requires a new layer inserted below transport. A NIRN client, on the other hand, can use NIRN with no changes whatsoever. Second, because NIRN uses native IP anycast, it is easier to derive proximity from NIRN than from i3. NIRN only has to measure distances between proxies—i3 has to measure distances to clients and targets. The main advantage of i3 over NIRN is that it is easier to deploy an i3 infrastructure than a NIRN infrastructure, precisely because i3 doesn't require IP anycast. Indeed, this has been a source of frustration for us—we can't just stick a NIRN proxy on Planetlab and start a service.

Regarding the broader indirection comparison, there is no question that i3 is more general. Its ability for both the sender or receiver to chain services is very powerful. The addressing space is essentially infinite, and hosts can create addresses locally. Finally the security model (which supports the chaining) is elegant and powerful. Having said that, NIRN does provide indirection from which benefits other than just anycast derive. For unicast communications, it could be used to provide mobility, anonymity, DoS protection, and global connectivity through NATs. In the best of all worlds, we'd want something like i3 running over NIRN. But IPv6 and NAT have taught us that you don't always get the best of all worlds, and considering NIRN's backwards compatility, it may after all be the more compelling story.

VI. ANYCAST APPLICATIONS

Given that NIRN offers an easy-to-use global IP anycast service which combines the positive aspects of both native IP anycast and application-layer anycast, it is interesting to consider new ways in which such a service could be used.

A. Peer Discovery

Though IP anycast has long been regarded as a means of service discovery, this has always been in the context of clients finding single servers. NIRN opens up discovery for P2P networks, where not only is there no client/server distinction, but peers must often find (and be found by) multiple peers, and those peers can come and go rapidly. Examples of such cases include BitTorrent and network games. Indeed, peer discovery has always been the achilles heel of P2P networks, since they typically rely on central rendezvous servers which may be overloaded. NIRN's two-tier architecture allows this to scale.

One reason that traditional IP anycast has not worked for peer discovery (other than difficulty of deployment), is that an IP anycast group member cannot send to the group—packets are just routed back to themselves. With the right selection characteristics, NIRN can support a wide-range of P2P applications. Random selection would allow peers to find arbitrary

[12]Note that the way i3 has described their anycast, it wouldn't scale to very large or very dynamic groups, because a single node holds all the targets and receives pings from the targets. It may be possible that i3 could achieve this with a model closer to how they do multicast, but we're not sure.

[13]for reasons described in first paragraph of section III-F

[14]they can be engineered to be fast by relying on IGP for convergence

other peers, and is useful to insure that unstructured P2P networks are not partitioned. Proximity is obviously also important, but to insure that a peer can find multiple nearby peers (rather than the same peer over and over), a selection service whereby a node can provide a short list of targets to exclude (i.e. already-discovered targets) would be needed.

B. Reaching an Overlay network

A very compelling application of NIRN would allow a RON [20] network to scale to many thousands of members, and would allow those members to use RON not only for exchanging packets with each other, but with any host on the Internet! What follows is a high-level description of the approach. Assume a set of 50-100 RON "infrastructure" nodes that serve many thousands of RON clients. The RON nodes all join a large set of anycast groups—large enough that there is an anycast transport address (TA) for every possible client connection. The RON nodes also partition the anycast TAs so that each TA maps to a single RON node. Clients discover nearby RON nodes (or a couple of them) using one of the anycast groups, and establish a unicast tunnel (for instance, a VPN tunnel) with the RON node. We call this the *RON tunnel*, and the RON node is referred to as the *local RON*.

When a client wishes to establish a connection with some remote host on the Internet, it does so through its RON tunnel. The local RON assigns one of its TAs to the connection using NAT, and forwards the packet to the remote host. When the remote host returns a packet, it reaches a nearby RON node, called the *remote RON*. Because the transport address of the return packet maps to the local RON node, the remote RON node can identify the local RON node. The remote RON tags the packet with its own identity, and transmits the packet through the RON network to the local RON node, which caches the identity of the remote RON, and delivers the packet to the client. Now subsequent packets from the client to the remote host can also traverse the RON network.

This trick isn't limited to RONs. It could also work for route optimization in Mobile IP[15] (for v4 or v6, see [41] for a description of the problem), or simply as a way to anonymize traffic without sacrificing performance.

VII. Other Features

In this paper, we have presented the basic aspects of NIRN. A "practical" IP anycast service, however, requires a number of features that we don't have space to describe in detail. Here, we outline some of those features, and show that the NIRN model is flexible enough to incorporate them.

A. Security

The IP routing infrastructure is secured router-by-router through human supervision of router configuration. This makes it quite difficult to masquerade as another host with another address (and not incidently, is much of why it is hard to deploy IP anycast).

NIRN, on the other hand, needs to explicitly secure its *join* and *leave* primitives. For this purpose, NIRN could adapt any of a number of network or wireless authentication protocols, including EAP and IPsec. The NIRN architecture could support other ways to control group membership. For example, at group creation time, an authenticated target could specify a list of hosts allowed to join the group which would then be stored at the RAP of the group as part of the group characteristics. Such a feature is very hard to deploy with native IP anycast.

The *join* and *leave* primitives also make NIRN susceptible to DOS attacks where proxies are flooded with such requests. Hence, it is important that the authorization process ensure the proxies do a small amount of work for every unit of work done by the target (or attacker). Using puzzle challenges would be one way to achieve this. The use of native IP anycast to reach the proxies means that an attacker is restricted to access NIRN services through a particular proxy. As a result, the impact of a DDoS attack is spread across the proxies.

B. Intra-domain operation

IP routing cleanly separates interdomain and intradomain routing, so that packets between hosts within a domain may always stay in that domain. This characteristic carries over to native IP anycast—as long as there is an anycast target in a given domain, that target will be chosen over those outside the domain. One way to replicate this characteristic with NIRN is to simply place a NIRN proxy inside the enterprise. Another would be to deploy native IP anycast within the enterprise, since deployment and scaling issues are not so severe in an enterprise. These approaches would not allow a client outside of the enterprise to discover a target inside the enterprise, but this would normally be the desired behavior. If not, then a *NIRN gateway* would need to be installed at the border of the enterprise. Joins inside the enterprise would be translated into joins outside the enterprise (using the unicast address of the gateway). Packets to an anycast group from outside the enterprise would likewise be tunneled or NAT'd to the internal target.

Perhaps a more important and common situation would be one whereby two targets were on the same LAN. This would be the case for instance at a gaming LAN party. It would obviously be nice if the game could automatically determine if other gamers were on the local LAN, and if not, then discover peers over the Internet. Without getting into details, it is easy to envision an approach whereby a NIRN TA could be algorithmically mapped into an IP multicast address and UDP port. NIRN targets would try this local IP multicast-based discovery before falling back on global NIRN.

VIII. Conclusion

In this paper, we propose a proxy based IP anycast service that addresses most of the limitations of native IP anycast. Specifically, the primary contribution of this paper is the design of NIRN, a practically deployable IP anycast architecture. The unique features of NIRN such as the scalability by the size and dynamics of groups mean that it opens up new avenues of anycast usage. The purported scalability has been substantiated through simulations representing extreme, but real, workloads. Simulations on the real tier-1 topology of the Internet point to the efficiency of our approach. Preliminary measurements across $\sim$26000 hosts and the j root-server show that a naive native IP anycast deployment might not offer good latency-based

[15] Details withheld for lack of space.

proximity; however, we present a deployment strategy by which this could be alleviated.

The fact that NIRN uses native IP anycast means that it can be used as a simple and general means of discovery and bootstrapping. Internet measurements against the anycasted DNS root-servers show that the reliance on native IP anycast does not undermine NIRN's ability to support connection oriented services. A NIRN prototype has been built and the deployment efforts are underway. We feel confident that NIRN has the potential of fulfilling the need for a generic Internet-wide anycast service which can serve as a building block of many applications, both old and new.

REFERENCES

[1] C. Alaettinoglu, S. Casner, NANOG TALK : "ISIS Routing on the Qwest Backbone: a Recipe for Subsecond ISIS Convergence", Feb 2002 (NANOG 24)

[2] C. Partridge,T. Mendez,W. Milliken , "HostAnycasting Service", RFC 1546, November 1993.

[3] C. Huitema, "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, June 2001

[4] D. Kim,D. Meyer,H. Kilmer,D. Farinacci, "Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP)", RFC 3446, January 2003

[5] T. Hardy, "Distributing Authoritative Name Servers via Shared Unicast Addresses", RFC 3258, April 2002

[6] R. Hinden, S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003

[7] http://www.as112.net/

[8] D. Katabi, "The Use of IP-Anycast for Building Efficient Multicast Trees", Global Internet'99, Costa Rica, 1999

[9] J. Abley, "Hierarchical Anycast for Global Service Distribution", ISC Technical Note ISC-TN-2003-1, http://www.isc.org/tn/isc-tn-2003-1.html

[10] D. Katabi, J. Wroclawski, "A Framework for Global IP-Anycast (GIA)," ACM SIGCOMM 2000

[11] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, "Internet indirection infrastructure," ACM SIGCOMM, August 2002.

[12] D. Adkins, K. Lakshminarayanan, A. Perrig, I. Stoica, "Towards a More Functional and Secure Network Infrastructure", UCB Technical Report No. UCB/CSD-03-1242, 2003

[13] E. Basturk, R. Haas, R. Engel, D. Kandlur, V. Peris, and D. Saha, "Using Network Layer Anycast for Load Distribution in the Internet," Global Internet 1998.

[14] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, N. Shah, and Z. Fei, "Application Layer Anycasting," Proc. IEEE INFOCOM'97 (1997).

[15] Z. Fei, S. Bhattacharjee, M. H. Ammar, and E. W. Zegura, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service," Proc. IEEE INFOCOM98 (1998).

[16] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", ACM Symposium on Theory of Computing'97

[17] http://www.planet-lab.org/

[18] http://www.isc.org/index.pl?/ops/f-root/

[19] N. Spring, R. Mahajan, and T. Anderson., "Quantifying the Causes of Path Inflation", SIGCOMM 2003

[20] D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, "Resilient Overlay Networks," ACM SOSP, Canada, October 2001

[21] K. Sripanidkulchai, A. Ganjam, B. Maggs, H. Zhang,"The feasibility of supporting large-scale live streaming applications with dynamic application end-points", ACM SIGCOMM 2004

[22] http://www.netfilter.org

[23] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", ACM SOSP 2003

[24] A. J. Ganesh, A. Kermarrec, L. Massoulie, "SCAMP: Peer-to-peer lightweight membership service for large-scale group communication", In Proc. 3rd International Workshop on Networked Group Communication (NGC), 2001

[25] J. Abley, "A Software Approach to Distributing Requests for DNS Service Using GNU Zebra, ISC BIND 9, and FreeBSD", Freenix Track, USENIX'04

[26] Z. Mao, R. Govindan, G. Varghese, R. Katz, "Route Flap Dampening exacerabtes Internet routing convergence", SIGCOMM'02.

[27] A. Gupta, B. Liskov, and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays", HotOS-IX, May2003.

[28] F. Dabek, R. Cox, F. Kaashoek, R. Morris, "Vivaldi: A Decentralized Network Coordinate System," SIGCOMM '04.

[29] T. Eugene Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", INFOCOM'02

[30] http://www.akamai.com/en/resources/pdf/whitepapers/ Akamai_Internet_Bottlenecks_Whitepaper.pdf .

[31] R. Rodrigues,B. Liskov,L. Shrira,"The Design of a Robust Peer-to-Peer System", ACM SIGOPS European Workshop,Sep. 2002.

[32] S. Ata, H. Kitamura, M. Murata, "Applications of IPv6 Anycasting", WORK IN PROGRESS - draft-ata-ipv6-anycast-app-00,

[33] P. Barber, M. Larson, M. Kosters, P. Toscano, "Life and Times of J-Root", Nanog Presentation, http://www.nanog.org/mtg-0410/kosters.html

[34] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, Oct'02

[35] S. Deering, W. Fenner, B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, Oct'99

[36] http://www.ripe.net/ripe/maillists/archives/routing-wg/2004/msg00183.html

[37] L. Subramanian, S. Agarwal, J. Rexford, R. H. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points", IEEE Infocom, June 2002

[38] K. Gummadi, S. Saroiu and S. Gribble., "King: Estimating Latency between Arbitrary Internet End Hosts", SIGCOMM IMW, Nov'02

[39] B. Greene, D. McPherson,"ISP Security: Deploying and Using Sinkholes", NANOG talk, http://www.nanog.org/ mtg-0306/sink.html

[40] http://www.ssfnet.org/homePage.html

[41] "Mobility for IPv6 (mip6)", IETF Working Group Charter, http://www.ietf.org/html.charters/mip6-charter.html

[42] "Scaling the Internet through Tunnels," Submitted to Sigcomm 2005 (author names withheld for anonymity)