Towards a deployable IP Anycast service

Hitesh Ballani, Paul Francis
Cornell University
{*hitesh,francis*}*@cs.cornell.edu*

Abstract

Since it was first described in 1993, IP anycast has been a promising technology for simple, efficient, and robust service discovery, and for connectionless services. Due to scaling issues, the difficulty of deployment, and lack of application-specific features such as load balancing and connection affinity, the use of IP anycast is limited to a small number of critical low-level services such as DNS root server replication. More commonly, application-layer anycast, such as DNS-based redirection, is used. As the number of P2P and overlay services grows, however, the advantages of IP anycast become more appealing. This paper proposes a new proxy overlay deployment model for IP anycast that overcomes most of the limitations of native IP anycast. We believe that this makes IP anycast a viable option for easing deployment and improving the robustness and efficiency of many P2P and overlay technologies. We describe the new deployment model, some of its uses for P2P and overlay networks, its pros and cons relative to application-layer anycast, and discuss research issues.

1 Motivation

IP anycast is an IP addressing mode (v4 or v6) whereby multiple geographically disperse hosts are assigned the same IP address, with the result that IP routing delivers packets destined to the address to the nearest[1] such host [1]. This works without any changes to unicast routing as routers do not distinguish between multiple routes to multiple different hosts and multiple routes to the same host.

There are three broad uses for IP anycast: *service discovery, query/reply services, and routing services*. With service discovery, IP anycast routes the client's packets to a nearby server, which then redirects the client to a server (possibly itself) which is subsequently accessed using IP unicast. With query/reply services, the entire exchange is done using IP anycast. With routing services, IP anycast routes the client's packets to a routing infrastructure (eg. IP multicast), which then continues to forward the packet using whatever technology is appropriate.

Together these constitute a powerful set of tools that can ease configuration, and improve robustness and efficiency for many applications or lower-layer protocols. There are three primary reasons for the simplicity and the power of by IP anycast: First, it operates at a low level, depending only on the IP routing substrate. This makes it robust, scalable for large anycast groups (though not for large numbers of groups), and simple for clients to use (once it is in place). Second, it automatically discovers nearby resources, eliminating the need for complex proximity discovery mechanisms [21]. 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.

Examples of IP anycast routing services include routing IP multicast packets to shared multicast tree rendezvous points [3, 6] and routing IPv6 packets (tunnelled over IPv4) to IPv4/IPv6 transition devices [2]. The only wide-scale deployments of IP anycast in a production environment are query/reply services for DNS: transparently replicating the root DNS servers [7, 5], primarily to spread load as a defense against DDoS attacks, and establishing "sink-holes" for DNS PTR queries to private addresses [14]. On a local scale, IP anycast is used by operators to simplify and improve local DNS server availability [15] as well as to establish sink-holes [15].

In spite of the power of IP anycast, there are several major problems that currently limit its use to a small number of critical applications like DNS root server replication. They include:

- IP anycast is incredibly wasteful of addresses. Because the routing infrastructure won't accept IP prefixes longer than a /22 (or a pre-CIDR /24), a single IP anycast group consumes 1024 (or 256) scarce IP addresses. The alternative would be to modify routing policy to accept larger prefixes, but that would open the door to huge routing tables, which leads us to the second problem:

- IP anycast scales poorly by the number of anycast groups. Each such group requires a BGP routing entry in the global routing system. GIA [8] proposed router modifications to improve scalability, but expecting core router upgrades for this purpose is almost certainly a non-starter.

- IP anycast is difficult and in some cases, impossible for users to deploy. It requires that the user obtain a block of IP addresses and an AS number, something that is currently outside the normal allocation policies of registration authorities (i.e. ARIN, RIPE, etc.). Even if such a block is obtained, each IP anycast destination (target) must run a routing protocol with the upstream ISP, which requires negotiations with the ISP, a significant amount of manual configuration, and a certain level of expertise.

- IP anycast is subject to the limitations of IP routing, in several ways. First, IP may suddenly route packets to a different anycast target, thus breaking the notion of connection affinity[2], which in turn breaks stateful protocols like TCP. Second, IP routing has no notion of load—not even link/router load, let alone server load. This problem is addressed in [19], but again by requiring changes in routers. Finally, BGP sometimes converges slowly, making a destination unreachable for many seconds or even minutes [16].

Because of these limitations, 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 (as they do not require any router modifications), provide fine grained control over target server load, and naturally maintain connection affinity. Because of these advantages, application-layer anycast is the method of choice for Content Distribution Networks (CDN) today.

Imagine for the moment that the shortcomings of IP anycast could be eliminated without sacrificing (at least not by much) its advantages. Were this possible, the uses for IP anycast would expand dramatically, especially for overlay and P2P technologies. For instance, IP anycast could be used to bootstrap members of a DHT (eg. Chord [11]), P2P multicast overlay [24], or P2P file sharing [25] network without requiring a central server to redirect joining members to existing members. Indeed, eliminating the bottleneck and single points of failure imposed by the central server(s) remains an open problem for P2P networks of all kinds [26]. This would work by having each member join the IP anycast group once it becomes a member. Subsequently newly joining members would transmit "member discovery" messages to the anycast group, thus discovering a nearby member.

Likewise, IP anycast could be used by clients of a DHT to simply and efficiently query the DHT, or to query services that themselves are built on DHTs, like DNS-style name resolution [27]. IP anycast could be used to send HTTP queries to nearby web proxies, without the need for explicit configuration of the web proxies or the latency overhead of a DNS query or an HTTP redirect.

IP anycast could be used to efficiently transmit packets into overlay networks like RON [12] or i3 [9]. RON is a particularly interesting case, as it would allow nodes that are not aware of the RON overlay to never-the-less use the RON overlay. The basic idea here is that all N members of a RON overlay would join an identical set of N anycast groups. The anycast address of each group would represent one of the RON nodes. Packets from a non-RON node J to a given RON node X would be routed via IP anycast to the nearest RON node Y. RON node Y could then forward the packet to X via the RON overlay. Likewise, return packets from X to J could be sent through the RON network to Y and then forwarded using unicast to J. This would greatly expand the scope of a RON network: from only being able to transmit packets between RON members to being able to transmit packets between RON members and any node in the Internet.

If the IP anycast service could be extended so that a node could be both a client and a target[3] (i.e., IP anycast packets sent by a member of the anycast group would be forwarded to the nearest group member *other than the sender*), then still more uses can be envisioned. For instance, networked game players could find nearby partners, and members of a P2P multicast overlay could find nearby peers.

2 Proxy IP Anycast Service (PIAS)

PIAS is an IP anycast deployment approach that overcomes most of the limitations of native IP anycast while maintaining most of its strengths. The basic idea is to implement IP anycast in an overlay, in much the same spirit as implementing IP multicast in the mbone overlay. Specifically, a large number of anycast proxies are deployed around the Internet. These are router-like boxes that advertise a block of IP anycast addresses into the routing fabric (BGP, IGPs), but are not themselves the anycast target destinations. Instead, packets that reach the anycast proxies through native IP anycast are subsequently tunnelled (or NATed) to the true target

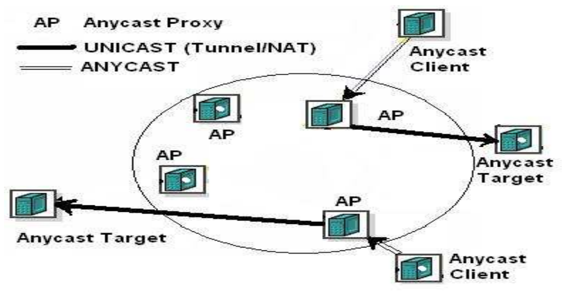

Figure 1: Proxy Architecture

destinations[4] using unicast IP (see figure 1). Hosts become anycast targets by registering with a nearby anycast proxy, which is itself discovered using native IP anycast!

The PIAS architecture solves the first three limitations of IP anycast cited above. It solves the problem of inefficient address usage because all the IP addresses in the prefix advertised by the proxies can be used by different anycast groups. In fact, PIAS does one better. It identifies an IP 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. Likewise, it solves the routing scaling problem by allowing so many anycast groups to share a single address prefix. Finally, it makes it very easy for a host to become an IP anycast target. All the host has to do is register with a proxy. There are no special routing requirements. The task of obtaining the address block/AS numbers falls upon the infrastructure operator; the effort put into this deployment is amortized across all the groups the infrastructure can support.

Of course, the reader may (and should) argue that all we've done is push the scaling and addressing problems from IP routing into the proxy overlay. This is very true, and quite intentional: the problems are much easier to solve when isolated from IP routing in this way. We now address scaling and other design issues in the proxy overlay. We start by stating our design goals:

- Scale by the number of groups
- Scale by the size of any group.
- Scale by group dynamics, by both continuous member churn and flash crowds, including those caused by DDoS attacks.
- Scale to $\sim 10^5$ proxies. 50 proxies in each of the largest 200 ISPs, which strikes us as plenty of proxies, gives us $\sim 10^4$ proxies; to be safe we target for an order of magnitude more.
- Backwards compatible, implying no changes at the clients and minimal changes (at least no network stack changes) at the targets[5].
- Offer features associated with application-layer anycast: load balancing, connection affinity, and the ability for a target to also be a client.

The first design goal dictates that we cannot require each proxy to know of all groups. As a result, for each group we designate a small number of proxies, called Rendezvous Anycast Proxies (RAP), to keep track of the membership of the group. We map groups to RAPs using consistent hashing[10], thus spreading the load of maintaining membership information evenly over the set of proxies. Each group is assigned a small number of replicated RAPs, for reliability reasons. Note that we require all proxies to know the status of all other proxies. This is justified by the maximum number of expected proxies (10^5) and the stable nature of the proxies. The proxies may maintain this global information through flooding, gossip [22] or a hierarchical structure [18]. Such an arrangement ensures that we attain a simple one-hop DHT and hence, limit the latency overhead of routing through the proxy overlay.

The RAP approach described above doesn't scale if a given group is very large, or has a lot of churn (second & third goals), since each of a small number of RAPs have to maintain membership for the whole group. A large group would risk overwhelming the RAPs with state information while a group with a lot of churn would lead to a stream of update information to the RAPs. Thus, we add another tier of membership management in the form of the Join Anycast Proxy (JAP). Specifically, the JAP is the proxy that is contacted by a target when it joins the group. The fact that the join is done through native IP anycast implies that the JAP is close to the target. The JAP is responsible for authenticating targets, and for maintaining liveness status of targets. The JAP also tells the RAP approximately how many targets it has for a given group (i.e. within 20% or 30% of the exact number). This way some number of targets can join or quit a given JAP without the RAP needing to be told. As a result, for very large groups, the RAP at worst scales according to the number of proxies, and for very dynamic groups the rate of updates to the RAPs is bounded and tunable.

This two tier JAP/RAP architecture results in packet paths as shown in Figures 2 and 3. When a client of the anycast service sends the first packet to the group, it reaches the Ingress Anycast Proxy (IAP) through native IP anycast routing. If the IAP does not know of any JAPs for the transport address (TA) this packet is destined to, the IAP resolves the TA to a RAP and tunnels the packet there. The RAP selects a JAP based on certain criteria (proximity to the IAP, load balance, or affinity, or some combination of these, as specified by the sixth goal) and tunnels the packet there. It also informs the IAP of the

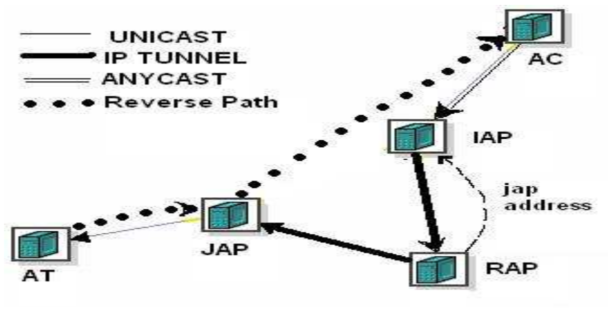

Figure 2: Initial packet path - 4 Segments long

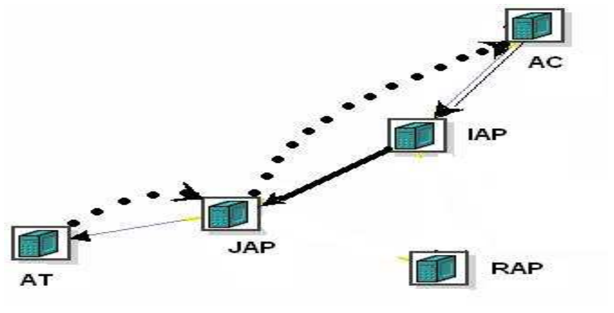

Figure 3: Subsequent packet path - 3 Segments long

selected JAP so that subsequent packets are tunnelled directly from the IAP to the JAP (Figure 3). Finally, the JAP sends the packet to the target, either by tunnelling if the target can de-tunnel packets, or using NAT otherwise (fifth goal)[6]. Note that since the IAP is close to the client, and the JAP is close to the target, the extra distance required to traverse those proxies typically not significant. We evaluated the overhead of this 3-segment path through simulations involving synthetic topologies generated using GT-ITM[23] and the results confirmed our intuition about the minimal overhead.

The reverse path transits the JAP, but avoids the IAP. The reverse path must go through at least one proxy, because the target is unlikely to be able to spoof the source address to be that of the anycast group, and the client expects a packet from the anycast group (fifth goal). By sending the return packet through the JAP and not the IAP, we allow the JAP to better (passively) monitor the health of the target, and to maintain NAT state. Note that it makes no sense for the IAP to try to monitor the health of the target, because packets may be flowing to the target through many IAPs, but through only one JAP.

Finally, the JAP is responsible for flushing the cached state from IAPs should it lose all of its targets (or enough of its targets that it should shed some of its load). Hence, if all the targets for a TA go down and the JAP receives a packet for the same TA, the JAP can send a control message to the IAP asking it to invalidate the cache entry. This forces the IAP to go back to the RAP for the group and ask for the address of some other JAP for the same group. As a result, the IAP is able to safely cache information for a long time. In case the JAP crashes, the IAP must learn of this and go to the RAP for subsequent packets. This would be attained using a health monitoring system[7] where each proxy's health is monitored by a small group of other proxies and a proxy's demise is disseminated using a flood through the overlay. The crashing of the JAP does not affect the target, whose packets will be anycasted to some other proxy, which takes up the job of the JAP for this target by asking it to register again.

2.1 Advanced Features

As stated above, the RAP may select the JAP based on a number of criteria, including proximity, connection affinity, and load balancing. The JAP subsequently selects a target, possibly also based on connection affinity and load balancing criteria. It is this selection process that imbues PIAS with features normally found only in application-layer anycast. As such, this aspect of PIAS deserves more discussion.

The first thing to point out is that these three 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 or the other criteria must be compromised. This is of course true of application-level anycast as well.

The second thing to point out is that the overlay structure of PIAS actually weakens its ability to find a target near a client, as compared to native "E2E" IP anycast. With PIAS, we know that the client is near the IAP, and the target is near the JAP (because both paths are discovered by native IP anycast), but anycast cannot be used to insure that the IAP and JAP are near each other. Therefore, the proxies must explicitly determine their distance to each other. While we haven't settled on the best way to do this, we note that scalable proximity addressing schemes like Vivaldi [20] provide one reasonable approach.

Because of the way we scale the RAPs (give them only "rough" information about the targets at a given JAP), and because we use multiple RAPs for each group, we cannot provide exact load balance for all groups (though we might be able to do so for a small number of select groups). Instead, we aim for "statistical" load balancing. Never-the-less, this is much better than what is provided by native IP anycast.

Finally, we note that proxies could potentially base their target selection on still other criteria. For instance, in the case where a target in a given anycast group is also a client of that group, proxies can exclude that target from their selection. A proxy could select a random target, something that might be useful for instance for spreading gossip. A proxy could use some kind of administrative scoping to select a target, for instance selecting a target with the same (unicast) IP prefix as the client. A proxy could even replicate packets and send them to multiple targets.

2.2 Implementation

We have implemented and tested the basic PIAS system in the laboratory as a sanity check for our ideas and to get a better grip on the implementation issues facing us. With the system geared towards router-like boxes, the current implementation has 2 components:
1. A user-space component responsible for the overlay management tasks, such as handling proxy join/leaves, target join/leaves, health monitoring etc.
2. A kernel-space component responsible for the actual forwarding of packets through the use of Netfilter hooks[13]. This involves tunnelling of the packets when sending them between 2 proxy nodes, and using a NAT[17] when handling packets to/from a target. The current kernel implementation doubles up the packet forwarding time as compared to the normal packet forwarding by the unmodified kernel. The actual figures are not presented as they do not offer any additional insight into the overhead involved.

While such an implementation is geared towards a scenario where we have our own infrastructure, piggybacking our deployment on an existing research testbed (e.g. RON [12]) would necessitate a pure user-level implementation. This would involve a not-too-difficult porting of the kernel level component to user space.

We feel that this implementation, although useful as a sanity check, does not have much to offer in terms of understanding the issues that we need to deal with (for an actual deployment). Hence, our main challenge, is deploying PIAS on the Internet and convincing applications to use our infrastructure service. For this purpose, 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.

3 Discussion and Research Issues

PIAS solves the major issues that limit IP anycast deployment. In doing so, it slightly weakens some of the major strengths of native IP anycast. In particular, we've lost some of the natural robustness of IP anycast, since we now rely on more than just IP routing. In addition, we've lost some of the simple nearness properties of native IP anycast. Nevertheless, we believe that PIAS is adequately robust and will provide good nearness properties in addition to the problems it solves.

Because PIAS uses indirection, it deserves comparison with i3 [9]. One major difference is that i3 requires changes to the network stack[8] to add the i3 layer between the current network and transport layers. PIAS requires no changes in clients whatsoever, and can operate with no changes in servers (if some surrogate node registers on its behalf). This makes deployment of PIAS easier than that of i3. A second important difference is that PIAS uses IP anycast, thus making it straightforward to derive proximity. Indeed, i3 could benefit from using PIAS, both as a means of i3 node discovery and routing packets to nearby i3 nodes.

One major problem that we haven't yet discussed is connection affinity. The issue is how to maintain affinity when native IP routing causes a different IAP to be selected during a given client connection (if the same IAP is always used, then the IAP will continue to use the same JAP that it initially cached). 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. This way, for instance, even if IP routing caused packets from a given client to select a different IAP, it could be routed to the same JAP. Unfortunately, this approach completely sacrifices proximity and load balance.

A better alternative would be to identify groups of IAPs among which any route changes are highly likely to take place. For instance, all IAPs in an ISP, a given POP, or a metro area. These IAPs could then coordinate in some way to provide affinity. How best to do this, or indeed determining if it is even necessary, will require experimentation.

Fortunately such complex affinity mechanisms may in fact not be necessary—the affinity provided by IP routing may in fact be good enough. We ran some measurements against existing anycasted DNS root servers and anycast sink holes to determine how often IP routing selected different destinations. We found that native IP anycast itself provides good affinity. Over 9 days of measurements at a rate of a probe every minute from 40 Planetlab sites to six anycast targets, 93.75% of the source-destination pairs never changed. The remaining 6.25% of the source-destination pairs (15 pairs) experienced a total of 120 route changes over the entire duration (i.e over $\sim$ 13000*15 probes), with at most 8 changes for any given source-destination pair. These experiments were also repeated with higher probe rates (once every 10 seconds) to make sure we were not missing out on very frequent flaps—the results appear similar.

To put this in more concrete terms, the probability that a two minute connection would experience a change is roughly 1 in 13,000, and the probability that a one hour connection would experience a change is roughly 1 in 450. If these numbers hold (or improve!) across a larger anycast deployment, then it is clear that most, though not all, applications would not require any affinity mechanisms beyond those provided by IP routing. This is one area that requires further experimentation.

A second main area for further research is how to run BGP and IGP routing so that both routing changes and routing failures are minimized, and so that routing selects good paths to proxies. There are concerns that the use of policies in inter-domain routing adversely impact it's ability to find nearby destinations. We tried to measure the stretch between anycast latencies and the shortest unicast path to the afore-mentioned DNS root servers and sink-holes, but found it difficult to get conclusive results, in part because of hierarchical nature of these anycast deployments[7]. On a bright note, measurements by operators of J-root[28], which is anycasted at 16 different locations, have shown that anycast does provide a decent amount of correlation between the position of the server and the clients that use it. The impact of BGP instability is one area where PIAS seems to be worse as compared to native IP anycast due to the addition of the proxies between the end hosts. Modelling the systems so as to rely on IGPs for faster convergence and using redundancy[9] to counter the occasional BGP event seem to be the only options to ensure that PIAS is not severely hit by these factors. Another concern is the impact of such a large anycast deployment on BGP dynamics. While we hope that the fact that the anycast prefix is being advertised from a large number of locations will lead to localized and rarer BGP hold downs [16], the opposite could occur.

While we do not have space to delve into the deployment model, some questions that need to be addressed for a practical service include the billing model, the security of the infrastructure, provision of a flexible and efficient interface to the customers and so on. There is also the issue of the number of proxies and their locations (in terms of number of tier-x ISPs and number of proxies in each POP for each selected ISP) so as to provide some form of load balance and to keep the overhead of traversing the proxy infrastructure limited.

Ultimately, the only way we can resolve these research issues is with deployment of a working PIAS and experimentations with real applications.

Notes

[1]nearest according to the routing metrics used by the routing protocols—this meaning holds throughout the paper

[2]tendency of subsequent packets of a "connection" to be delivered to the same target - referred hereon as affinity. While lack of affinity is perceived as a major anycast weakness, preliminary measurements discussed later show that this perception may be overly pessimistic

[3]in much the same way that a member of an IP multicast group can be both a sender and a receiver.

[4]members of an anycast group

[5]we would like to be able to support legacy server-side applications and are able to do so.

[6]Figures 2,3 assume that the JAP uses a NAT

[7]here we are concerned with the health of the proxies rather than the JAP keeping track of the health of one of its targets

[8]though this could be done through the use of a proxy on the end-host, allowing legacy applications to use i3

[9]using multiple anycast prefixes - each service is given more than one transport address

References

[1] C. Partridge et. al., "HostAnycasting Service", RFC 1546, November 1993.

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

[3] D. Kim et. al., "Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP)", RFC 3446, January 2003

[4] Y. Rekhter et. al., "A Border Gateway Protocol 4 (BGP-4)", RFC 1771, March 1995

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

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

[7] 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

[8] Katabi, D. et. al., "A Framework for Global IP-Anycast (GIA)," ACM SIGCOMM 2000

[9] I. Stoica et. al., "Internet indirection infrastructure," ACM SIGCOMM, August 2002.

[10] Karger, D. et. al., Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. ACM Symposium on Theory of Computing (El Paso, TX, May 1997).

[11] Ion Stoica et. al. "Chord: A scalable peer-to-peer lookup service for internet applications". ACM SIGCOMM, August 2001.

[12] David G. Andersen, et. al., "Resilient Overlay Networks," ACM SOSP, Canada, October 2001

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

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

[15] http://www.nanog.org/mtg-0310/miller.html

[16] Z. Mao et. al., "Route Flap Dampening exacerabtes Internet routing convergence", SIGCOMM'02.

[17] P. Tsuchiya et. al., "Extending the IP Internet Through Address Reuse," ACM SIGCOMM Computer Communications Review, 23(1):16-33.

[18] A. Gupta et. al., "One Hop Lookups for Peer-to-Peer Overlays", HotOS-IX, May2003.

[19] W. T. Zaumen et. al., "Load-Balanced Anycast Routing in Computer Networks", ISCC 2000.

[20] Frank Dabek et. al., "Vivaldi: A Decentralized Network Coordinate System," SIGCOMM '04.

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

[22] R. Rodrigues et. al.,"The Design of a Robust Peer-to-Peer System", ACM SIGOPS European Workshop,Sep. 2002.

[23] Ellen W. Zegura et. al. "How to model an internetwork", IEEE-Infocom, March 1996.

[24] Dejan Kostic, et. al., "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh" SOSP 2003

[25] P. Karbhari, et. al., "Bootstrapping in Gnutella: A Measurement Study", PAM'04

[26] M. Castro, et. al. "One ring to rule them all: Service discovery and binding in structured peer-topeer overlay networks", SIGOPS European Workshop, 2002.

[27] V. Ramasubramanian et. al., "The Design and Implementation of a Next Generation Name Service for the Internet", SIGCOMM, August 2004.

[28] P.Barber et. al., "Life and Times of J-Root", Nanog Presentation, http://www.nanog.org/mtg-0410/kosters.html