

# USING IP ANYCAST FOR LOAD DISTRIBUTION AND SERVER LOCATION

Robert Engel, Vinod Peris and Debanjan Saha  
{rengel, vperis, debanjan}@watson.ibm.com  
IBM T. J. Watson Research Center  
30 SawMill River Road  
Hawthorne, NY 10532

Erol Basturk  
basturk@pluris.com  
Pluris Network Systems  
10455A Bandley Drive  
Cupertino, CA 95014

Robert Haas  
rha@zurich.ibm.com  
IBM Zurich Research Laboratory  
Saumerstrasse 4  
CH-8803 Rueschlikon/Switzerland

## Abstract

An anycast address is an IP address that may be bound to one or more network endpoints. Unlike multicast, a packet destined to an anycast address is forwarded to *any one* of the hosts with this address. In this paper, we investigate how the IP anycast service can be used as an effective load distribution mechanism at the network layer. The anycast service naturally provides a client with the ability to locate the topologically closest server. We discuss changes required in the host protocol stack that enable applications to transparently use the anycast service. More specifically, we propose a scheme that limits the required changes to the IP layer processing and describe its implementation in the AIX TCP/IP stack. Finally, we consider an application example where anycast communication is used to distribute load among a set of mirror web sites.

## 1 Introduction

In the Internet environment there are several instances where clients need to locate services provided by the network. In order to increase service availability and provide load distribution, it is now common practice to replicate servers that are providing these services. Examples include WWW “mirror” sites, multiple SOCKS servers [20] and proxy servers. These replicated services can be classified into (1) local replicas, and (2) distributed replicas. Local replicas, such as those provided on a cluster of workstations, may be contained within a single subnetwork. On the other hand, distributed replicas could be geographically dispersed across the Internet. For clients to be able to locate the appropriate server it is necessary to extend and enhance the location mechanisms used for directing clients to these services.

In the Internet it is fairly typical for a server to be identified by a name as opposed to a network address. Naming mecha-

nisms such as the Domain Name System (DNS) [22] provide mappings from host names to their IP addresses. DNS can be used to map virtual host names (service names) to a set of IP addresses that correspond to the servers providing the replicated service [7]. Take the case of a client making a Web access with a single name that is associated with a set of HTTP servers. When the client sends a name resolution query the DNS server maps the name to any one HTTP server. Each successive query can be mapped to an IP address associated with a different HTTP server in a Round Robin manner. This will distribute the client requests to the different HTTP servers and share the load among the servers. Rather than a simple Round Robin scheme the resolution process may also be guided by network and server load. The Distributed Director [9] as well as the proposed Host Proximity Service (HOPS) [15] direct the client to the “closest” server, where distance is measured with respect to some metric based on network topology. Along the same lines [5] proposes an application layer anycast service to support replicated services. Both Distributed Director and HOPS servers interact with the routing protocols to pre-compute a measure of the distance of the client from each of the candidate servers. While these approaches have some attractive features, they disable the caching mechanism of DNS and therefore are likely to increase the DNS query load. Past studies have shown that DNS related traffic is not insignificant [10]. Moreover, in order to direct a client to the closest server, the name server has to obtain fairly accurate network topology information, which requires close interaction with the routing protocols.

An alternative approach to supporting replicated services is with the use of anycast addresses. An anycast address is an IP address that may be bound to one or more network endpoints. Different servers that are providing the same service can all have the same anycast address. A host is considered a member of an anycast group simply by virtue of the fact that the anycast address is bound to one of its interfaces. It is not necessary for any router in the network to be explicitly aware of the membership of an anycast group. A packet addressed to an anycast

address is delivered to one of the members of the anycast group. This approach is elegant in that forwarding of packets bound to anycast addresses is handled directly by the routers which have access to accurate network topology information.

Traditionally, the IP service is only limited to delivering datagrams to end-hosts without considering how the applications use them. In this regard the anycast service is no exception. However most if not all applications have some notion of state that links successive packets together. Clearly, if as a result of anycasting, successive packets are forwarded to different anycast servers this state information will be lost, rendering anycasting useless for all but a few instances that involve a single packet exchange. In order to ensure that anycast datagrams belonging to a single flow are not routed to different end-hosts we need to make some minor modifications in the host TCP/IP protocol stack. The basic idea is to pin the end-host to which the first packet of the flow has been sent. This is very similar to route pinning in the context of QoS routing [16]. The pinning is done by inserting a loose source route option in all subsequent packets from the same flow. We have implemented these modifications in the TCP/IP protocol stack of AIX 4.2 which is a variant of the 4.x BSD implementation.

In this paper we explore some techniques for providing anycast services in the Internet. In Section 2, we briefly define the anycast service that is part of the Internet standard and discuss some of the implications of forwarding anycast datagrams. Section 3 is devoted to the issue of ensuring that packets from the same flow are always routed to the same end host. There we describe both the transmit and receive modifications to the TCP/IP stack in AIX to ensure that stateful connections can be maintained between two end hosts. In Section 4, we discuss how anycast communication can be used to architect a scalable infrastructure for world-wide web. The paper ends with a brief conclusion that summarizes our contributions.

## 2 Anycast Communication in the Internet

The notion of anycasting in the Internet was first introduced by an RFC produced by the Internet Research Task Force [25]. The authors of [25] motivated the need for an anycast address by giving examples of clients trying to locate a server that is closest to them. They define IP anycasting as: “a service which provides a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address”. Anycasting is also defined in IP version 6 (IPv6) albeit with some minor differences [12] that are outlined below. In the following we clarify some of the issues that are part of the anycast service definition, namely: (1) identification of an anycast datagram, (2) forwarding of an anycast datagram, and (3) advertisement of anycast servers.

In both IPv4 and IPv6 an anycast datagram is no different from a regular IP datagram. The destination address field in the datagram corresponds to an anycast address. In [25] the authors suggest carving out the IP address space for a separate class of anycast addresses. They contend that this will reduce the risk of applications mistakenly failing to recognize anycast addresses. Apart from chewing up a significant amount of the IP address space which is already a scarce resource, this approach places an additional burden on the routing protocols to dis-

tribute reachability information related to these new addresses. The special anycast addresses will not aggregate with the regular unicast routing information and so it might result in excessive amount of route advertisements. In contrast, IPv6 specifies that anycast addresses are identical to unicast addresses, with the difference that it allows multiple hosts (more specifically, network interfaces) to be registered with the same address.

A router that receives an anycast datagram might have multiple next hops over which it can forward this datagram. It is recommended that the router picks any one of these possible next hops to forward this datagram. Since it is possible for unicast IP packets to be duplicated, it is possible for an anycast datagram to be delivered to multiple hosts. However, the specifications [12, 25] make it clear that the intended behavior is for anycast datagrams to be delivered to a single host.

Clearly, members of an anycast group have to indicate to the routers that they wish to receive anycast datagrams for a specific address. One approach is to use an enhanced version of the Internet Group Management Protocol (IGMP) so that hosts can advertise this information to routers. This will be particularly useful on shared media LANs so that all the routers on the same media will get this multicast information. Each router can then advertise its reachability to a host with an anycast address, through regular routing updates with its neighbors. Another approach is to have the end hosts run a routing protocol so that they can directly update the routers with the information about which anycast addresses they serve. Note that the hosts do not need to run the full blown routing protocol; rather they only need to be able to advertise reachability to the anycast addresses that they serve. A third and simpler approach is to statically configure the routers with the information about the different hosts that are serving an anycast address.

### 2.1 Forwarding Anycast Datagrams

One of our main objectives in this paper is to use the anycasting mechanism to effectively distribute traffic to several different servers. To understand how a router can distribute load, let us consider a scenario where multiple servers sharing the same anycast address are reachable via a single router. From a routing perspective these servers appear as a multipath route to a single destination. Now, given that the router has a multipath route to a particular address – the router cannot distinguish between unicast and anycast addresses – what is an appropriate way to distribute traffic among these multiple routes? Broadly speaking there are two ways in which this can be done: (1) stateful approach, and (2) stateless approach. The idea behind stateful forwarding is to identify the flow that each packet belongs to, so that packets from the same flow can be directed to the same end-host. This method requires the router to keep track of the active flows that are passing through it. The stateless approach on the other hand does not require any state to be maintained at the router; the router simply forwards each packet independently.

The *Connection Router* approach described in [13] is a good example of a stateful forwarding scheme. A Connection Router maintains a list of all the active flows, typically TCP connections, that are passing through it. When an IP packet has to be forwarded, the Connection Router first checks to see if the pro-

protocol type indicates that it is TCP. If not, it just forwards the packet on any one of the alternative routes to its destination. Otherwise, it proceeds to search its list for the corresponding flow (identified by the source address/port number and destination address/port number). A successful match yields the next IP hop (route) that the packet has to be sent to. A packet belonging to a new TCP flow will fail this search, resulting in the addition of this TCP flow along with the associated next hop information to the list of active flows. The determination of which next hop to choose for a particular flow is a local policy decision.

Note that this mechanism requires the router to maintain a significant amount of state information for all the flows that are being routed through it. Routers typically are optimized for forwarding packets. Modifying routers to maintain flow states as well as the classification of the incoming IP packets into flows is likely to impact their efficiency. Also, this approach works well only when the replicated servers are topologically close to the Connection Router since all incoming traffic is routed through the Connection Router. Another drawback of the Connection Router is that it only applies to TCP flows and so cannot be used to distribute load when the traffic is UDP.

In the case of stateless forwarding the router treats each packet on an individual basis. This is the typical behavior of a router, wherein it forwards each packet using one of several possible routes without regard to maintaining consistency between the successive packets that belong to the same flow. For the purpose of load distribution, we consider three simple schemes in our study, namely: (1) round-robin, (2) random, and (3) hash-based. Assume that a router has an equal-cost multipath route to a certain destination. In the first scheme, the router forwards successive packets addressed to that destination in a round-robin manner, while in the second it randomly picks one of the multiple routes to forward the packet. Note that both of these schemes can result in successive packets from the same flow being forwarded to different next hops. Consequently, anycast packets belonging to the same flow might end up at different end hosts, which is clearly not of much use to the application, especially when it is “flow-based”, e.g. telnet, HTTP, SMTP, etc. In Section 3, we describe modifications to the host protocol processing that alleviates this problem by adding a source route option in the packet in order to pin an anycast flow to a unique destination.

Alternatively, one can use a hash-based approach to consistently route all the packets in a flow to the same host. If there are multiple next hop candidates for a given destination address, the source address/port number in the packet can be used as a key to selecting one of them. Assuming that there is no routing transient, this ensures that packets from the same flow are sent out on the same next hop at each of the routers along the way. To ensure that all the packets of a flow are routed to the same end host, all the routers along the path must support this feature and forward packets belonging to the same flow consistently. Another caveat is that a change in the multipath route at a particular router may lead to a change in the outcome of the hash function. Consequently, packets of the same flow may take different paths. This could result in a disruption of all the flows that share a multipath route. Typically, route changes do not occur very frequently in the Internet, and as result would have

a small impact on short-lived connections [26, 8]. They could however be more disruptive to long-lived connections. As we will see in Section 3, pinning a flow to its destination eliminates these potential disruptions as well as the need for consistent forwarding at intermediary routers along the path.

When multiple hosts sharing a single anycast address are connected to the same network segment it is the responsibility of the router to distribute packets destined to this anycast address. It is possible to enhance the ARP [27] (in IPv4) or Neighbor Discovery [24] (in IPv6) mechanisms at the router to maintain all link-layer addresses that map to the shared address. One of the policies described in Section 2.1 can then be used to forward the packets that are destined to the anycast address. Only the router to which the hosts sharing the anycast address are connected, needs to implement these changes in ARP or Neighbor Discovery. An alternative to modifying link-layer mechanisms is to use routing mechanisms either through configuration at the router or by allowing the hosts to participate in the routing protocols. More details about both these mechanisms can be found in [3].

### 3 Maintaining Stateful Connections

From the previous discussion it is evident that the anycasting service does not require routers to maintain any state. In other words the network has no obligation to send two successive datagrams destined to the same anycast address to the same end station. This may be a significant problem for applications that maintain stateful connections to remote hosts using anycast addresses. In this section we discuss modifications to the host protocol stack to ensure that anycast packets belonging to the same flow are treated in a consistent manner.

In [25] Partridge et al. suggest a modification to TCP that would allow all packets belonging to a TCP connection to be delivered to the same host even when the destination address is an anycast address. In their solution, when a host initiates a TCP connection to a remote anycast address, it treats the destination address as a wildcard address in its local protocol control block, but uses the anycast address as the destination address in the first packet of the TCP connection (SYN packet). When the SYN packet is received by one of the hosts in the anycast group, it responds by sending a SYN-ACK where it uses one of its unicast addresses as the source address. Upon receiving the SYN-ACK, the initiating host instantiates the remote address of the connection in its local protocol control block with the unicast address of the remote host and uses this address as the destination address in subsequent exchanges. This ensures that after receiving the SYN-ACK the TCP connection is bound to a single remote host.

The problem with the scheme described above is that it requires changes to the TCP semantics. Protocol stacks at the initiator of the connection, as well as the recipient, have to be modified in order to realize such changes in the TCP semantics. In a client/server model, both client (initiator) and server (recipient) must be changed, which is not desirable in an infrastructure such as the world-wide web, where the number of clients is much larger than the number of servers.

Another approach suggested in [28], in the context of multihoming can also be adapted to bind an anycast address to a uni-

cast address. The idea here is to use a new IP option, named Source Identification Option, to inform clients that a particular communication initiated through the use of an anycast address should proceed with the use of the unicast address of one of the anycast group members. In this scheme, the initiator of the connection sends the datagram to the anycast address that is ultimately received by one of the group members. In its reply, the remote host puts one of its unicast addresses (the remote host may be multi-homed) in the source identification option. Upon receipt of the reply, the initiator can replace the anycast destination address with the unicast address supplied by the remote host. The drawback of this solution is that it requires a new IP option to be defined.

We propose an alternative approach to maintain the association between an anycast address and the unicast address of one of the hosts in the anycast group for the duration of a connection. The advantage of this approach is that it uses an existing IP option, namely Source Route Option, available in both IPv4 and IPv6 and requires minor modifications to protocol processing. Furthermore, this modification impacts only the recipient of a TCP connection, making it suitable to a large client/server environment such as the world-wide web, where only the servers need to be changed.

In the following we describe the proposed solution and its implementation in AIX 4.2 TCP/IP stack, which is a variant of the 4.x BSD implementation.

### 3.1 Modification to the Transmit Path

We consider the case where TCP is used as the transport protocol. Assume that a host with a unicast address  $C$  is initiating a TCP connection to a remote address  $A$ , that happens to be an anycast address. The initiating host first sends a SYN packet with  $A$  as the destination address. The network routes the packet to one of the hosts in the anycast group. The TCP module of the remote host processes the SYN packet and generates a SYN-ACK with  $S$  as the destination address and  $A$  as the source address and passes that on for processing by the IP layer. The IP layer determines the appropriate interface on which the packet should be forwarded. It also recognizes that the source address of the packet is an anycast address. At this point, it inserts a source route indicating that the packet has been forwarded through one of the unicast addresses of the outgoing interface, say  $U$ . Note that the check to determine if an address is an anycast address and insertion of the source route is a deviation from standard IP layer processing at the host. When the SYN-ACK is received at the other end, it appears to have arrived from  $A$  via  $U$ . The IP layer processes the source route and passes that on to the TCP layer, which stores it in the TCP control block. In subsequent exchanges to the remote host, the source route is reversed and used to route the packets. This causes the packets to be routed through the unicast address of the remote host and establishes a binding between the anycast address  $A$  and the unicast address  $U$  for the lifetime of the connection.

Figure 1(a) explains the scheme described above in more details. It shows the state variables, namely the source and destination addresses, and the source route at different layers of the protocol stacks at both ends of the connection. Note that

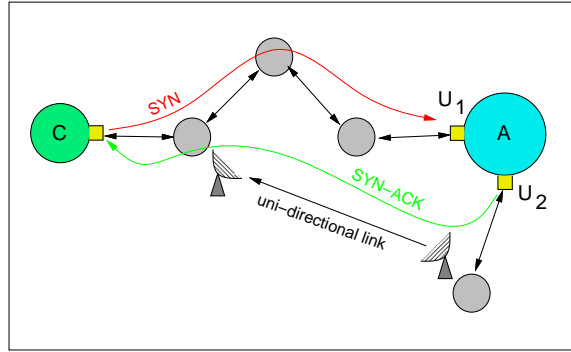


Figure 2: Asymmetric Routing and multi-homed host.

while the state variables are part of the TCP state for the life time of the connection, at the IP layer they are part of the state associated with the current packet being processed. As shown in Figure 1(a), the source route is inserted by the IP layer at the remote host while processing the SYN-ACK packet. Note that the source route consists of only one hop, the unicast address of the interface. Hence, IP option processing has to be performed only at the end-host, not at the routers on the path. The filled triangle on the right reflects the position of the next hop pointer in the source route option field in the IP header and indicates that this hop has already been traversed. When the SYN-ACK reaches the destination, the IP layer processes the source route and passes it on to the TCP layer where it becomes a part of the TCP connection state. In the packet that follows, the TCP layer reverses the source route (it is a list consisting of a single hop in this case) and passes that on to the IP layer along with the source and destination addresses. The IP layer determines that the next hop for the packet is  $U$  and puts that as the destination address. It also puts  $A$ , the ultimate destination, in source route option field in the IP header. Note, that the filled triangle is on the left of  $A$ . This indicates that the next hop to be traversed is  $A$ . When this packet is received at the remote host, as a part of standard source route processing it should swap  $A$  and  $U$  and place the filled triangle past  $U$  indicating that this hop has been traversed. It should then forward the packet to the final destination, which happens to be the same host. In Figure 1(a) we do not show the last step. It is in fact an optimization that saves us from a second round of processing at the IP layer. The IP stack at the remote host has been modified to recognize that the ultimate destination is itself. Consequently, it processes the packet and passes the source route to the TCP layer which stores it in the connection state. In subsequent exchanges, the TCP layer at the remote host reverses the source route and passes that on to the IP layer to be put into the IP header.

### 3.2 Modification to the Receive Path

Although the solution described above works in most cases, it may lead to problems when the remote host is multi-homed and routing is asymmetric. To understand this problem, let us consider the scenario pictured in Figure 2 where the remote host has two interfaces with unicast addresses  $U_1$  and  $U_2$ . In addition it also has an anycast address  $A$ . We assume that routing

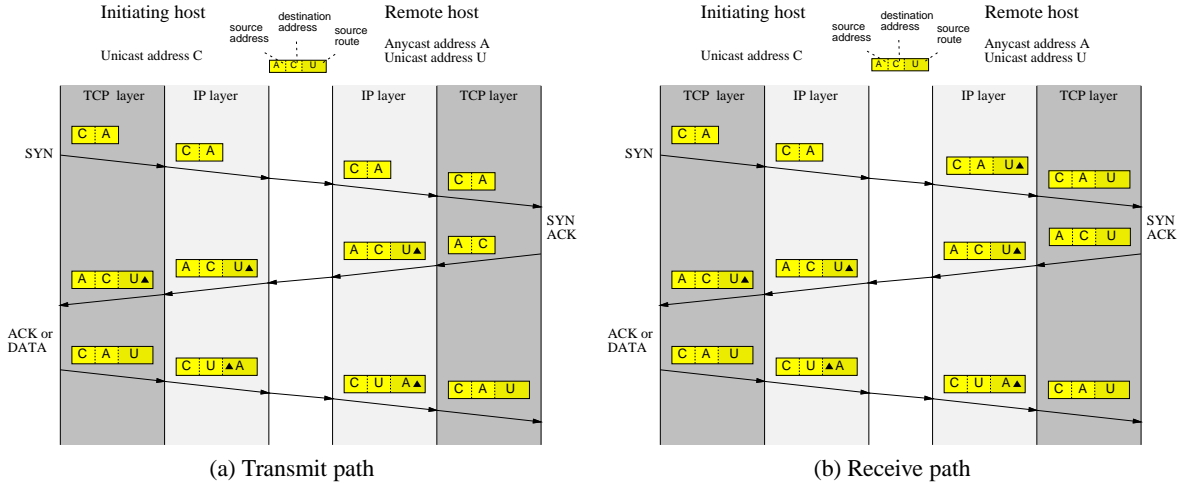


Figure 1: Source route insertion in the transmit and receive path.

is asymmetric and packets from  $C$  to  $A$  are routed via interface  $U_1$  while packets from  $A$  to  $C$  are routed via  $U_2$ . Now let us trace through the connection setup phase between  $C$  and  $A$ . As in the above description, the connection is initiated by a SYN packet with source address  $C$  and destination address  $A$ . In response, the TCP module at  $A$  generates a SYN-ACK with  $A$  as the source address and  $C$  as the destination address. The IP layer at the remote host traps the packet and recognizes that  $A$  is an anycast address and attempts to add a source route to indicate that the packet has been routed through a unicast address. The dilemma now is which one of  $U_1$  and  $U_2$  to use as the unicast address. The obvious choice is the interface which routing chooses as the outgoing interface for the packet destined to  $C$ . In our example  $U_2$  is the outgoing interface for destination  $C$ . If  $U_2$  is used as the intermediate hop, when  $C$  reverses the source route for subsequent exchanges with  $A$ , the packets are routed through  $U_2$  instead of  $U_1$ . However,  $U_1$  happens to be the route of choice for packets generated by  $C$  and destined to  $A$ . If routing is asymmetric and  $U_2$  is unreachable from  $C$ , the connection may be terminated.

An easy fix for this problem, is to insert the source route when a packet with  $A$  as the destination address is received at the remote host. As a part of IP layer processing, the remote host can insert a source route with the incoming interface  $U_1$  as the intermediate hop and pass that on to the upper layer. The upper layer then can store the source route in the connection control block and use the reversed route in subsequent exchanges. Care has to be taken to make sure that the source route is inserted only when appropriate. That is, if the packet already contains a source route with one of the unicast addresses of the host as the penultimate hop, no changes are required. For packets already carrying a source route option but where the penultimate hop is not one of the unicast addresses of the host, it has to be properly inserted in the source route.

Figure 1(b) shows the processing involved in the TCP and IP layers of the source and destination in more details. In this case the SYN packet is trapped at the IP layer in the remote host. It recognizes that the destination address in the packet is

one of its anycast addresses and inserts a source route with a single hop  $U$ . Note that the filled triangle is placed past  $U$  indicating that this hop has already been traversed. This source route is passed on to the TCP layer which stores it as part of the connection state. When the TCP layer at the remote host generates the SYN-ACK in response to the SYN packet, it passes the reversed source route to the SYN packet, it passes the reversed source route to the IP layer along with the source and destination addresses. The IP layer adds the source route in the header and forwards the packet. The source route processing at the IP layer presented here is a deviation from the source route processing in a standard IP stack and represents an optimization used in our implementation. In a standard implementation, the packet will pass through the IP layer twice, first time with  $U$  as the destination address and  $C$  as the next hop, and the next time with  $C$  as the destination address. In our implementation, these two passes are integrated into a single pass through the IP layer. When the SYN-ACK is received at the destination, the IP layer processes the source route and passes that on to TCP. The TCP layer saves the source route in its connection state and use the reversed route in subsequent exchanges.

In the discussion above we assume that TCP is the transport protocol. Although UDP does not have the notion of a connection, applications do use UDP as the underlying transport protocol for stateful communication. An application using UDP as the transport protocol has two modes of operation. It can either bind a transport layer end-point (*socket*) to a specific remote destination address and send data to only that address, or use the same end-point to send data to multiple remote addresses. Similarly, an application can use the same socket to receive data from multiple sources. Appropriate handling of one-to-many and many-to-one semantics of UDP sockets requires an enhancement to the solution described above. Additional details on the support for UDP sockets can be found in [3].

We have implemented the changes described in this section in the AIX 4.2 TCP/IP protocol stack. We have added a feature that allows one to configure an address as an anycast address. This is in essence an enhancement to IP aliasing feature sup-

ported by most TCP/IP stacks. Using the IP aliasing feature, one can specify one or more aliases for an interface. However, an anycast address is more than a simple alias. A distinction between the aliased unicast address and an anycast address is required since packets with an anycast address as a source or destination address are trapped by the host identified with that address and processed differently. As explained above, changes are required only at the servers, that is, hosts with anycast addresses. On the receive side the changes are contained within the `ipintr` function and on the transmit side changes are restricted to the `ip_output` function.

## 4 A Case Study

In the previous sections, we discussed possible enhancements to routing and host protocol processing that allow widespread use of network layer anycast without modification to the existing applications and their communication semantics. In this section, we focus on applications of anycast to provide improved network services. We consider world-wide web (www) as an example, and show how anycast can be used as a mechanism for load distribution and service location.

### 4.1 Scaling the Web

Exponential increase in traffic volume at popular web sites has forced site administrators and network service providers to use innovative means to improve the scalability of the web infrastructure. A commonly used technique to improve scalability is server replication. For popular web sites, replicas are placed at different strategic locations to exploit user locality. Each replica often consists of a cluster of server nodes which additionally increases the scalability and reliability of the server. Server replication however, leads to new problems. For the users, the problem is to find the best mirror site. For the network, the problem is to distribute the load among different replicas. A number of schemes have been suggested for server location and load distribution. In the following, we discuss different approaches to handle this problem and argue that network layer anycast can evolve to be a more scalable and potentially better solution.

One of the commonly used techniques for server location and load distribution is to use enhanced versions of Domain Name Service. A Domain Name Server (DNS) can resolve the same name to different IP addresses for the purpose of load distribution. Round Robin DNS and application layer anycasting [5] are examples of this scheme. The problem with these schemes is that they are not capable of determining the availability of a given server and continue to send client requests to failed servers. Since intermediate name servers cache the resolved name-to-IP-address mapping, changes in DNS information propagates slowly through the Internet. Even if a network administrator detects a failed server and removes its DNS records, the Internet as a whole may not become aware of this fact for hours or possibly days. Consequently the failed server continues to receive requests, resulting in HTTP failures to end-users. Caching of name-to-IP-address mapping by intermediate name servers also makes fine grain load distribution difficult. It is possible to time-out cached mappings quickly. However, that has several undesirable side effects. It increases the load

on the DNS server and network significantly. It also means that each (when caching is disabled) HTTP request to the web server has to be preceded by a DNS query to the name server. The other major problem with DNS based schemes is that they fail to factor in network topology and load in making decisions. Consequently, a client may be directed to a far away server or to a server attached to congested network even when one close to it is available.

The proposed Host Proximity Service (HOPS) [15] and the Distributed Director [9] alleviate a part of this problem by participating in routing protocols and by using other mechanisms to monitor server and network loads. However, they require the clients to consult the HOPS server (or the distributed director), each time a new connection is initiated. This essentially adds an extra round trip time to the HOPS server to the connection setup time. Given that most of the connections to web servers are relatively short, this overhead may significantly reduce the advantages of using this service.

For fine grain load balancing in a server cluster, many popular web sites use a connection routing front-end (also called connection router) that distributes connections to different server nodes. It maintains a view of the load on each server in order to forward new connections to the most appropriate server. One can combine a DNS based scheme with connection routing [13] front-ends in order to get the best of both approaches. However, this does not solve the problem of locating the closest server for the requesting client. Additionally, since all connections to a server cluster have to pass through the connection router, it is a single point of failure and may become a bottleneck at high loads.

We argue that network layer anycast provides a more scalable solution to the server location and load distribution problem as compared to the schemes described above. In an anycast based scheme, all mirror sites of a server share a single anycast address. When a client sends a connection request to the anycast address of the server, it automatically gets routed to the nearest mirror site. Unlike HOPS and DNS based schemes, network layer anycast service is a part of the routing infrastructure. It requires no additional mechanisms to determine the mirror site closest to a requesting client. It not only helps locate the closest server, but also distributes load among the replicas without introducing new servers (e.g. HOPS server, Distributed Director). It does not add any extra load on the networks in terms of DNS or HOPS queries. The clients also save a round trip time to the HOPS server or DNS. Since routing systems are much faster than the domain name service in detecting failures and responding to it, network and server failures have only temporary impact on the anycast based server location and load distribution scheme. Additionally, it has no single point of failure or bottleneck as is the case for a connection router. The deficiency of an anycast based scheme, as compared to a connection router, is that it cannot distribute load based on precise information on server load. An anycast based scheme, like DNS based schemes, distributes load randomly. Although a random distribution has the potential to cause a load imbalance, our experience so far has been to the contrary. In fact, our investigations indicate that with a little sophistication, network layer anycast can be as effective as a connection router for fine grain load distribution without the overhead of maintaining per-connection

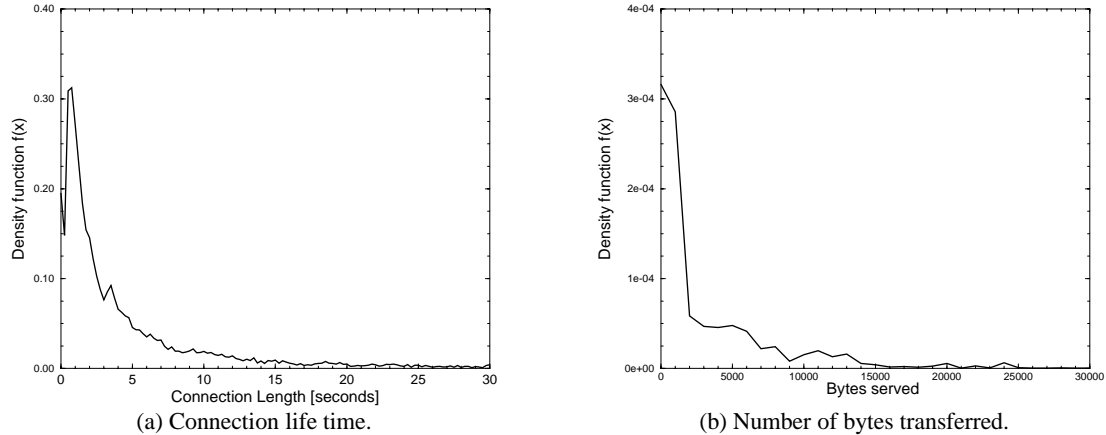


Figure 3: Distribution of connection life time and number of bytes transferred.

state. In the following we present some simulation results in support of this claim.

#### 4.2 Random Distribution of Load

To demonstrate fine grain load distribution using anycast, we use an experimental setup where  $N$  server nodes share a single anycast address. In the specific experiment reported in this paper,  $N$  is 5. We assume that the servers implement the modified protocol stack which pins a connection to a server node. Note that as far as the router is concerned, only the first packet of a connection (TCP SYN) has the anycast address as the destination address. The routers treat the route leading up to the anycast address as a multi-path route consisting of five different routes. We consider 3 different dispatching policies: (1) round-robin, (2) random, and (3) hash-based that the router can use to distribute load among the server nodes. In the round-robin scheme, packets destined to anycast address are forwarded to one of the routes in a round-robin manner. In a random dispatching scheme, the packets are forwarded to one of the randomly chosen routes. In the hash-based scheme, the route is chosen by hashing on the source address and the port number.

We use the *tcpdump* traces collected from the Official 1996 Olympic Web Server [2] as the traffic profile for this experiment. Traces were collected for approximately 2 weeks in 20-minute segments. The entire trace consists of approximately 60 million HTTP requests, from about 725,000 clients. In figure 3 we plot the distribution of connection holding time and amount of data transferred for a 4-minute segment from the trace. As seen in the figure, most of the connections are short-lived and transfer small amounts of data. This observation is representative of similar observations from other segments of the trace and is consistent with traces collected at other web sites.

Figure 4 shows the relative performance of different policies in distributing load. We compare the three policies described above with a near-ideal policy where a new connection is dispatched to the server with least number of active connections. This policy is used by many connection routers as a heuristic approximating the ideal load distribution policy. The simula-

tion keeps track of the number of active connections as well as the total bytes transferred at each of the 5 servers. The number of bytes transferred in half second intervals as well as the number of active connections are sampled every half second. Figure 4(a) plots the probability density function of load imbalance in terms of number of connections. In this figure, the X-axis is the difference between the maximum and minimum number of active connections (over all the 5 servers), normalized to the number of active connections in a sampling interval averaged over the 5 servers. The Y-axis is the probability density of maximum load imbalance among different nodes. Clearly, the closer the density function is to the Y-axis the better is the load balancing. As expected, the least active connection policy performs the best in this case. It manages to keep the load imbalance within 5% of the average load in the worst case. The performance of other policies are similar to each other. The load imbalance never exceeds 50% of the average load and is below 30% in most cases.

Although the number of active connections is a good measure of server load, the amount of data transferred is a more appropriate metric for network load. In Figure 4(b) we compare the four policies in terms of volume of data transfer. In this figure, the X-axis is the difference between the maximum and minimum number of bytes transferred in the last 0.5 second, normalized to the average number of bytes transferred by a node during the same time. Here, we observe that all four policies show very similar behavior. In this case, the maximum load imbalance is within 125% of the average load in most instances. However, there are points in time when load imbalance shoots over 300% of the average load. Fortunately, such cases are very rare. From these results, we conclude that in terms of network load distribution, anycast-based schemes perform as well as load distribution schemes that maintain information about connection states.

#### 4.3 Pragmatics of Using Source Routing

Our approach to use IP anycast service for server location and load distribution makes use of source routing. Source routing

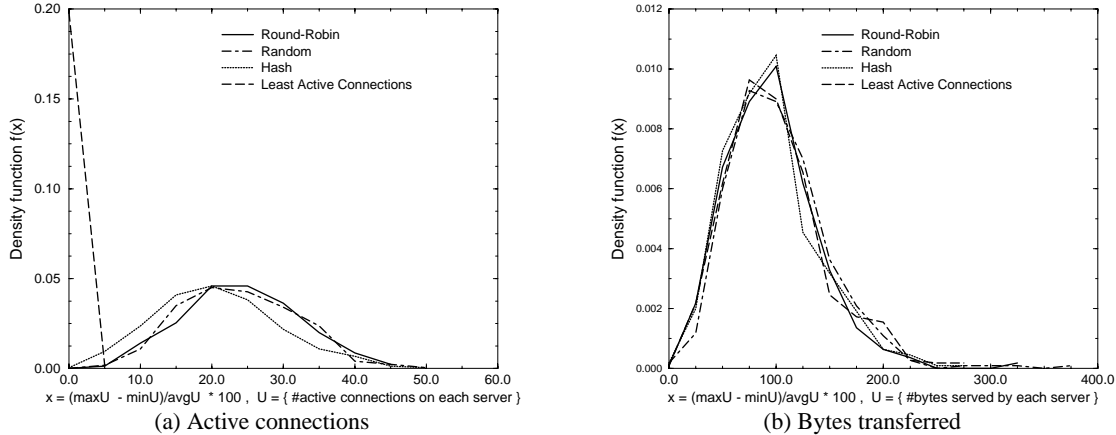


Figure 4: Distribution of number of active connections and bytes transferred

offers flexible means to provide useful services in the Internet. There are, however, security and performance implications of using source routing. In the absence of other security measures, source routing makes it easier to spoof the source of an IP packet. A malicious user can divert the packets that are destined to a particular host by inserting a source route through itself. However, source routing is not the only way to divert packets to a malicious host; the propagation of incorrect routing information or the use of ICMP redirect messages are examples of other possible attacks [4]. In fact it is well understood that reliance on the source address in a packet for any form of authentication can be extremely dangerous [4]. Currently, the issue of security in the Internet is being widely discussed in the scientific community and there are several standardization efforts in place at the IETF. End-to-End security can be provided both at the IP layer [1] as well as at the transport layer [14]. In conjunction with any of these security mechanisms source routed packets provide the same level of security as regular IP packets. Today, the Secure Socks Layer (SSL) is the most widely used end-to-end security mechanism in the Internet [29]. In fact, all secure web servers use SSL to avert potential security risks. SSL makes use of public key cryptography to authenticate and exchange keys between the communicating end-points. After the initial handshake all data is encrypted using a negotiated cipher scheme. The addition of source-routed packets does not in any way compromise the encryption and authentication capability that is provided by SSL.

Another potential problem with source routing is the overhead of processing IP options at the intermediate routers. This has been a problem with IPv4 since options can be inserted in a packet in any possible order. Hence, even if none of the options present in a packet are relevant to an intermediate router, it cannot determine this without parsing the entire list of options. To improve performance some routers treat packets with options with a lower priority than packets without options. This problem however has been corrected in IPv6 [12]. IPv6 defines a specific extension header for source routing. An intermediate router is not required to look at this extension header unless the destination address in the packet matches one of its inter-

faces. The solution proposed in Section 3 inserts a source route that only includes the end host. Thus in IPv6 only the end host is required to examine the routing header and the intermediate routers are not impacted in any way.

## 5 Conclusion

In this paper, we examined different aspects of using IP anycast as a mechanism for load distribution and service location in the Internet. We proposed a scheme that allows applications to transparently use anycast services without any change in their communication semantics. The proposed scheme does not require any modification to the routers and routing protocols. However, to exploit the full potential of anycast, certain enhancements to multi-path forwarding and/or address resolution are desirable. We described these enhancements along with some recommendations on how advertisements for anycast addresses can be distributed using the existing routing protocols. On the host side we proposed modifications to the protocol processing to ensure that packets belonging to a single flow are consistently routed to the same end-host. These modifications are limited to changes at the IP layer of the recipient of the TCP connection, making this scheme suitable to a client/server environment. We have implemented the proposed solution in the AIX 4.2 TCP/IP stack. Using a trace based simulation, we also evaluated several anycast-based load-distribution policies. Based on this study, it is fair to conclude that IP anycast should be considered seriously as a candidate mechanism for load distribution and service location in the Internet.

## References

- [1] R. Atkinson. RFC 1825: Security architecture for the Internet Protocol, August 1995.
- [2] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz. Analyzing stability in wide-area network performance. In *SIGMETRICS'97*, Seattle, June 1997.

- [3] E. Basturk, R. Engel, R. Haas, V. Peris, and D. Saha. Using network layer anycast for load distribution in the internet. Technical Report RC 20938, IBM Research Division, August 1997.
- [4] S. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, April 1989.
- [5] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proceedings of the IEEE INFOCOM '97*, 1997.
- [6] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5:47–76, February 1987.
- [7] T. Brisco. RFC 1794: DNS support for load balancing, April 1995.
- [8] B. Chinoy. Dynamics of internet routing information. In *Proceedings of SIGCOMM'93*, pages 45–52, September 1993.
- [9] Cisco distributed director. White Paper, 1996. Cisco Systems.
- [10] P. Danzig, D. Delucia, and K. Obraczka. An analysis of wide-area name server traffic. In *Proceedings of ACM SIGCOMM'92*, Baltimore, MD, August 1992.
- [11] P. Danzig, D. Delucia, and K. Obraczka. Massively replicating services in wide-area internetworks. Technical report, University of Southern California, 1994.
- [12] S. Deering and R. Hinden. RFC 1883: Internet protocol, version 6 (IPv6) specification, January 1996.
- [13] D. Dias, W. Kish, R. Muhkerjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of the IEEE Computer Conference (COMPCON)*, Santa Clara, March 1996.
- [14] T. Dierks, P. L. Karlton, A. O. Freier, and C. Kocher. The TLS protocol, version 1.0. `draft-ietf-tls-protocol-03.txt`, May 1997. Work in progress.
- [15] P. Francis. A call for an internet-wide host proximity service (hops). URL: <http://www.ingrid.org/hops/wp.html>, 1996.
- [16] R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. QoS routing mechanisms and OSPF extensions. `draft-guerin-qos-routing-ospf-01.txt`, March 1997. Work in progress.
- [17] J. Guyton and M. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of ACM SIGCOMM'95*, pages 288–298, 1995.
- [18] C. Huitema. *Routing in the Internet*. Prentice-Hall, 1995.
- [19] J. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.
- [20] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. RFC 1928: SOCKS protocol version 5, April 1996.
- [21] G. Malkin. RFC 1388: RIP version 2 carrying additional information, January 1993.
- [22] P. Mockapetris. RFC 1035: Domain names — implementation and specification, November 1987.
- [23] J. Moy. RFC 1583: OSPF version 2, March 1994.
- [24] T. Narten, E. Nordmark, and W. Simpson. RFC 1970: Neighbor discovery for IP Version 6 (IPv6), August 1996.
- [25] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, November 1993.
- [26] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, Computer Science Division, University of California, Berkeley, April 1997.
- [27] D. Plummer. RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on Ethernet hardware, November 1982.
- [28] M. Shand and M. Thomas. Multi-homed host support in IPv6. Internet Draft, June 1997.
- [29] Secure Socks Layer 3.0 specification. URL:<http://home.netscape.com/eng/ssl3/index.html>. Netscape Communications Corporation.
- [30] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. RFC 2165: Service Location Protocol, June 1997.