Three Practical Ways to Improve Your Network

Kevin Miller – Carnegie Mellon University

ABSTRACT

This paper presents three simple techniques for improving network service using relatively unknown features of many existing networks. The resulting system provides greater reliability, enhanced security, and ease of management. First, it addresses the application of IP anycast to provide reliable recursive DNS service. Next, it explains the use of unicast reverse path forwarding and its usefulness in preventing local nodes from originating packets with spoofed source addresses. Finally, it explains how unicast reverse path forwarding can be used to quickly and easily apply source address filters on your network. As an added benefit, some of these features provide mechanisms to conform a network to Best Common Practices (BCP) of network operators.

Anycast DNS Service

Anycast [1] is an IP addressing technique where unicast IP addresses are assigned to multiple hosts and routes configured accordingly. Routers receiving packets destined for anycast addresses select one of potentially several valid paths to hosts configured with the address. This technique can be used wherever unicast IP routing exists, as anycast IP addresses are simply unicast addresses designated by network operators. Here, we use anycast addressing to improve the reliability of DNS service, load balance DNS requests across a number of servers, minimize service downtime due to maintenance, and automatically direct requests to the topologically nearest server.

DNS Issues

Outages

Administrators of recursive DNS servers are well acquainted with the volume of complaints that arise when DNS services are unavailable. The domain name system is certainly a critical piece of network infrastructure in networks large and small. When clients lose their bridge to the system, network access quickly suffers, as nearly every client network application uses DNS to identify its server's IP address. Additionally, network servers may have several opaque uses of DNS during normal processing. Thus, while clients may not need DNS service after a connection is established, the server might require DNS service to respond to client requests.

Implementations of DNS resolver libraries typically allow the configuration of more than one DNS server. This seemingly provides a means of coping when a DNS server is unresponsive. Unfortunately, these failover mechanisms are very rudimentary on most operating systems, leading to extended outages when servers are unavailable.

The most popular mechanism for using alternate DNS servers is a mechanism we'll call the 'object impermanence' method. This method is characterized by the following behavior. Upon receiving a request for DNS resolution, the system identifies the primary configured DNS server and sends the request to this server. The resolver waits a predetermined period of time for a response, ranging from 0-5 seconds in our evaluation. Note that the resolver distinguishes between a negative response (no answer available) from the primary server and a total lack of response. In the former case, this result is propagated to the requesting application.

If no response is received within the timeout period, the request is sent to the next server on the server list and the timeout period begins anew. If the server list is exhausted without receiving a response, the system may consult internal tables (such as 'hosts' files) or return error information to the client. When the next DNS resolution request is received from an application, the system again starts by consulting the first DNS server on the list, even if it has provided no answer to previous queries. Thus the term 'object impermanence': no history of failed requests is kept. When the primary server is unavailable, each DNS resolution request is met with a delay equal to the response timeout, at minimum.

Further delays are added by operating systems' IPv6 resolution mechanisms. With widespread IPv6 implementations appearing in mainstream releases, many system resolvers perform queries for the IPv6 address of a hostname prior to requesting the IPv4 address [2]. Some systems have IPv6 address lookups enabled by default, while other systems perform this resolution after one-time configuration. Today, many IPv6 requests quickly return with no results. The resolver then retries the IPv6 lookup using each item of the domain search list appended to the query in succession. Once these queries fail, the original query is retried as an IPv4 address query.

Systems with the object impermanence behavior try each specific query against the primary server and

wait the response timeout period before querying the backup servers. Thus, we formulate a general expression for the extra delay added when the primary server is unavailable (assuming the second server is available and IPv6 resolution is enabled):

$$ExtraDelay\ (secs) = [Query\ Timeout] \times ([Searchlist\ Length] + 2)$$

While the delay to retrieve a single DNS response may seem tolerable, a single application operation often involves more than one query. Many applications perform both a forward lookup to retrieve an address for a hostname plus a reverse lookup to retrieve the hostname associated with the IP address. Some application configurations have further DNS requirements. For example, the SSH daemon using local password authentication requires only a single DNS request to login. However, when authenticating against our central Kerberos database and obtaining AFS access credentials, a single login skyrockets to requiring 10 DNS resolutions. Using the expression above, the extra delay added to an SSH login on our campus when the primary DNS server is unavailable is 50 seconds.

The behavior of DNS resolvers was tested on eight operating systems, and the results of these tests are in Table 1 below. Seven of the eight systems exhibit the object impermanence behavior. The most popular response timeout is one second (five systems use this value), though two systems have a longer five second timeout. Cisco IOS, on the other hand, sends queries to all configured servers simultaneously. Windows XP is the only system that retains some knowledge of unreachable servers. After a one second timeout on the first request, XP uses reachable servers immediately on subsequent queries. Over time, the dead servers are re-tested and once reachable, are used in the configured order.

Server Address Changes

An additional problem faced by DNS operators is the difficulty of moving DNS servers. Since, by definition, recursive DNS servers must be configured by IP address, when the DNS server IP address changes each client must update its server address list. The widespread use of DHCP helps minimize the difficulty of changing these addresses, but there are inevitably cases where DNS server information is statically configured. Before the use of anycast addresses at Carnegie Mellon, server transitions could take upwards of a year to complete as clients with old configurations were identified and contacted.

The use of anycast DNS servers virtually eliminates the hassle of changing DNS server addresses. As new servers are added and old ones removed, traffic is automatically redirected by routers and clients are unaware of the particular server answering their DNS requests. Additionally, an enterprise with multiple locations could elect to use the same anycast addresses in all locations, further reducing the complexity of DNS server configuration. Anycast servers could be located in each location for low-latency resolution, backed up by central servers. In fact, proposals have been floated at the IETF to designate specific IPv4 addresses as well-known anycast recursive DNS addresses, to further reduce host configuration requirements.

The Solution

Introduction

Using anycast addressing techniques and host-based routing daemons, we dramatically increase the reliability of recursive DNS service. This reliability is achieved by eliminating the dependence upon a single machine to answer requests to a specific IP address. Specifically, multiple servers that provide identical DNS services are each configured with the anycast IP address on an interface. When using these techniques, a single unreachable host is of little consequence as routers redirect all requests to working servers.

Designating Addresses

The first step in configuring this service is the designation of certain unicast IP addresses as anycast addresses. We recommend designating a small subnet for use as anycast addresses. While this subnet is not attached to any router interface, individual addresses within the subnet are used to provide a particular service. Specifically, it is undesirable for any client to believe the anycast address is on the same subnet as itself. This would lead to the client not forwarding requests to the first-hop router and defeat the principal strengths of the system. It is important the subnet is identified well, though, as one of the benefits of anycast DNS comes from the unchanging nature of the

Operating System	Impermanence?	Response Timeout
Cisco IOS 12.1(13)E6	Yes	0 sec
FreeBSD 5.1	Yes	5 sec
Linux 2.4.20	Yes	1 sec
Mac OS X 10.2.6	Yes	5 sec
OpenBSD 3.3	Yes	1 sec
Solaris 8	Yes	1 sec
Windows 2000-SP3	Yes	1 sec
Windows XP-SP1	No	1 sec

Table 1: Operating system DNS resolver behavior.

anycast service address. Once the subnet is identified, individual IP addresses within the subnet are allocated for use with a particular anycast service.

Server Configuration

Each server that participates as a member of the anycast service pool must have the designated anycast address configured on a local interface. Typically, a new virtual loopback interface is added and the anycast address is assigned to this interface. This provides the greatest flexibility to operators, especially if the host has multiple physical interfaces. If the anycast address is assigned as a secondary address of a physical interface, it would be unusable if the physical interface was inactive.

The DNS server software must also recognize and listen to network requests on the anycast address. In most cases, this is as simple as starting the server software after the address has been configured. Depending upon the configuration, though, the address may need to be explicitly listed. In environments where anycast is the only valid method of querying the DNS servers, administrators may wish to specifically reject requests on the non-anycast addresses. This prevents any accidental reliance upon the unique unicast address of the server.

Enabling Routing

Once the servers are configured to service DNS requests using the anycast IP addresses, the routers must be setup to direct packets to one of the anycast servers. The simplest such configuration is to directly configure the anycast addresses as static routes on each server's first-hop router. Each static route must then be distributed through the intra-domain routing protocol. The static routes are added as /32 (''host'') routes with the next-hop address being the unique address on the server's physical interface.

While static routing does provide anycast service, further enhancements can be made. Specifically, by running a host-based routing daemon on each DNS server, the entire system is more robust and reliable. In this case, each server announces a route to its anycast address(es) using the intra-domain routing protocol. This route is then propagated through the routing domain, subject to the constraints of the IGP. Routers within the domain will select the best path to a server announcing the anycast address.

The primary benefit of using a host-based router is the dramatically reduced response time from a server becoming unreachable. In the case of static routes, some DNS requests would continue to be forwarded to the unreachable server until the static route is manually removed. Using a host-based router, the first-hop router will identify (by the lack of protocol exchange) when the server is unreachable. At that time, routes originated by the host will be dropped from the routing table, causing routers to forward traffic to other working servers.

Benefits and Limitations

While anycast is a good solution for providing recursive DNS service, it is not a panacea for reliability of all network services. Because routers may have multiple choices for the path of anycast-destined packets, sequential packets may follow different paths and arrive at different hosts. Therefore, anycast is best used only for stateless protocols, such as DNS, Kerberos, and syslog. If a protocol provides a mechanism for discovery of unique addresses within the first packet exchange, it may be suitable to use anycast for load balancing and reliability. However, it is also assumed that the overhead to service each request is roughly equivalent; that is, there is little need to balance traffic based upon server load or usage.

The configuration described here assumes equal cost weighting of network routes to the various servers. This implies that requests will automatically be forwarded to their nearest working server. When the path to multiple servers is equivalent, equal cost multi-path routes will be used if supported by the router and routing protocol. In this case, the router will choose a route for each packet from the best routes. Administrators could also use unequal weightings on anycast IP routes to control the parameters by which certain servers are used.

This document addresses the use of anycast within an administrative domain. Inter-domain anycast is also in active use, as well. The ''F'' root server, for example, is using anycast addressing to distribute DNS root server load among several servers [3]. Additionally, it is used to setup DNS root servers in multiple locations around the globe, reducing resolution time for clients in areas without a high concentration of root servers.

Deployment Example

The deployment of anycast recursive DNS service at Carnegie Mellon was completed in December, 2002. To begin, we identified a range of addresses (within IP space in our control) that we would use for anycast-enabled services: 128.2.1.0/26. We then identified two addresses within this range for anycast recursive DNS service: 128.2.1.10 and 128.2.1.11. Each site deploying anycast services should take care to designate addresses within IP blocks in their control.

The system was configured on our four existing DNS servers, but clients were slowly transitioned to use the anycast service addresses. The entire process was completed with only a few problems that were quickly resolved. Due in part to the high reliability of BIND 9 [4], since February we've had continuous anycast DNS service. This is done while having staggered monthly reboots of our servers and physically relocating one machine. A basic diagram of our anycast DNS service is in Figure 1 below. The unique addresses of the DNS servers are listed beneath the anycast address in the figure.

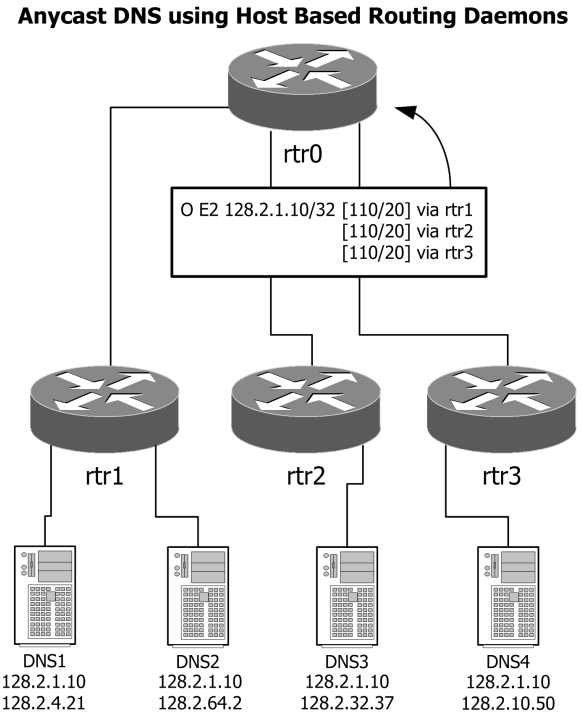

Figure 1: Four DNS servers answer requests to 128.2.1.10 .

The two problems encountered were simply part of the process of understanding the parameters necessary to keep the system continuously available. The first issue was with an ingress filter on the router connecting one DNS server's alternate link. The system worked until the dual-homed machine's default route changed, sending outgoing packets over the second link. The router ingress filter had not been updated with the anycast service addresses and was thus blocking traffic from the server. The other issue that administrators must remember is that if the name server is stopped, the router daemon must be shutdown or the anycast routes removed. In practice, this has been done manually as we've had no cases of the name server stopping except when requested. An automated mechanism for stopping the router when the name server is unresponsive would be beneficial, though.

Host Configuration

Each host that is a member of the anycast service pool is configured with the anycast service addresses as additional loopback addresses. On our Linux servers, these addresses are assigned to interfaces named 'lo:1' and 'lo:2', which are initialized and configured at boot time. The configuration of these interfaces is shown in Figure 2 below.

Machines with multiple interfaces to diverse networks have additional configuration requirements. Namely, the host's routing table is (as expected) used to direct IP packets out a given interface. Typically devices are configured with a single default route. In normal operation, this means that packets might be received on both interfaces (as our host-base router announces the anycast routes through both interfaces). However, most responses will follow the default route through one interface. This is perfectly functional, except when the interface or next hop of the default route is unreachable. Because the default route is statically configured, the machine will continue to send traffic through the now-useless interface.

The solution is to remove the static default route from the kernel routing table and allow the host-based routing daemon to keep the kernel default route fresh. If the originator of the best default route goes away, the default route from the alternate interface is used instead. Another problem arises, though, when the routing daemon is not running. In this case, there is no default route in the kernel routing table and an accompanying degraded network connection. Our solution to this problem lies within a utility started at boot time that monitors the kernel's default route. When the routing daemon is functioning, it removes any static default routes and records them for future use. If the routing daemon is stopped, the original default routes are re-added to the kernel routing table.

In named.conf:

```
options {
    listen-on { 128.2.1.10; 128.2.1.11; };
};
```

Figure 3: Limiting nameserver query access.

Service Configuration

Because our existing DNS servers were re-used as anycast DNS hosts, we did not explicitly remove access to the unique server addresses. Thus, no changes were necessary to the configuration of our nameserver, BIND 9. New servers added to the anycast service pool, however, are configured to only accept queries on the anycast addresses. This prevents accidental use of the host's unique IP address and makes it possible to easily remove hosts without any worry that client machines will suffer. The configuration fragment to limit the query access in BIND 9 is shown in Figure 3. Administrators should substitute the site anycast DNS addresses for our addresses in the fragment.

```
lo:1      Link encap:Local Loopback
          inet addr:128.2.1.10  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1

lo:2      Link encap:Local Loopback
          inet addr:128.2.1.11  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

Figure 2: Configuration of Loopback interfaces in Linux.

Routing Configuration

While our IGP is OSPF [5], the techniques used to configure and announce the anycast host routes are similar in other routing protocols. We chose the Quagga [6] (descendent of Zebra) routing daemon as our host router due to its open source license and support for OSPF. Additionally, Quagga configuration files are similar to Cisco IOS configurations, a format we are accustomed with. Quagga operates with one central daemon process that maintains interface and global system configuration parameters. Supported routing protocols run in a separate daemon process for each protocol. In our case, then, each DNS server runs two processes: the 'zebrad' central process, and the 'ospfd' process to associate with the backbone routers. A home-grown script reads basic configuration parameters of an anycast host and generates appropriate configuration files at boot time. This makes it very easy to configure additional anycast addresses as necessary.

The anycast addresses are configured as static host routes in the main Quagga configuration. The OSPF configuration enables physical interfaces in an OSPF NSSA area and redistributes the static anycast DNS routes into the OSPF area. A redistribution filter provides protection against announcement of unintended routes through the server. Fragments of the configuration files are shown in Figure 4 below.

The use of OSPF Not-So-Stubby Areas [7] (NSSAs) is deliberate. In some cases, our anycast DNS servers are multi-homed for additional redundancy and network locality. However, we explicitly do not want the servers used as a path between backbone routers. Server interfaces are configured in separate NSSA areas and hear only a default route from the backbone area. The relevant configuration parameters for the upstream Cisco routers in our network are shown in Figure 5.

Finalizing Deployment

After configuring each server and the upstream routers, we verified that the hosts and routers are properly exchanging OSPF routing information. Additionally, we verified the valid host routes for the anycast addresses and that servers were correctly answering requests to the anycast addresses. Once complete, we updated the DNS server parameters assigned by DHCP to clients and announced the new server addresses.

In quagga.conf:

```
interface eth0
 ip address 128.2.4.21/26
!
interface lo:1
 ip address 128.2.1.10/32
!
interface lo:2
 ip address 128.2.1.11/32
```

In ospfd.conf:

```
interface eth0
 ip ospf authentication message-digest
 ip ospf message-digest-key 1 md5 [key]
!
router ospf
 ospf router-id 128.2.4.21
 ospf abr-type cisco
 compatible rfc1583
 area 128.2.4.0 authentication message-digest
 area 128.2.4.0 nssa
 network 128.2.4.21/26 area 128.2.4.0
 redistribute connected
 distribute-list 50 out connected
!
access-list 50 permit host 128.2.1.10
access-list 50 permit host 128.2.1.11
```

Figure 4: Quagga configuration file fragments.

Upstream Cisco Router:

```
router ospf 1
 area 0.0.0.0 authentication message-digest
 area 128.2.4.0 authentication message-digest
 area 128.2.4.0 nssa default-information-originate no-summary
 network 128.2.4.0 0.0.0.63 area 128.2.4.0
 network 128.2.0.0 0.0.255.255 area 0.0.0.0
```

Figure 5: Cisco Router configuration.

Summary

IP anycast can substantially improve the reliability of recursive DNS service and ease server management tasks. When multiple DNS servers are configured with host-based routing daemons, the system's responsiveness to server or network failure can be measured in mere seconds. Given the current client resolver implementations, a fast response to such failures dramatically reduces the DNS resolution delays. Anycast is a remarkably easy technology to deploy, and we believe that can be successfully used in many environments.

Unicast Reverse Path Forwarding

One feature emerging in router hardware and software is unicast reverse path forwarding (''uRPF''). Enabling uRPF on every interface of the network edge improves the security of the entire Internet through source address verification. Though it requires almost no operator effort to enable, many operators are unaware of its benefits. Source address verification prevents machines from sending packets with clearly forged source addresses, quickly stopping some classes of network attacks. It also minimizes the difficulty in tracking other denial of service attacks, as packets have validated source addresses. In short, there are compelling reasons that proactive networks should use unicast reverse path forwarding.

Need for Source Address Verification

As the popularity and reach of the Internet continues to grow, the number of individuals actively seeking methods to exploit vulnerabilities in the infrastructure rises as well. While this is good for the long term security of the network, sometimes the fundamental principles upon which we operate networks are challenged. One of these challenges has been to build mechanisms for verifying the source address of IP packets being forwarded through a router. Until denial of service (DoS) and distributed DoS (DDoS) attacks began occurring with regularity, there was little incentive to develop widespread and easy mechanisms to verify a packet's source address.

While distributed denial of service attack strategies use diverse methods of achieving their goal, some of these strategies involve generating IP packets with a source address other than an accurate address of the attacking host. Some attacks, such as the Smurf [8] attack, set the source address to the attack target. Using IP directed broadcasts, a single attack packet can cause thousands of machines to respond to the faked source address. The Smurf attack thus employs an amplifier to make the attack more potent. Other attack strategies involve simply randomizing the source address field to make an attack harder to trace back to a specific point of access to the network.

As the frequency of these attacks rises, so does the interest in encouraging the implementation of source address verification (SAV) in routers. This led to network operators and the IETF publishing Best Common Practice #38 [9] in May 2000, following growing concerns after the Teardrop [10] and Smurf attacks began in late 1997. This document recommends that network operators implement SAV mechanisms on the edge of the network using traditional 'access list' methods of filtering packets. At the same time, work began on developing automated methods of implementing SAV.

Verification Mechanisms

The basic question in applying address verification is what source addresses are expected to be inbound to the router on a particular interface. This is information that the network engineers might know but must be able to express to the routers in convenient ways. BCP38 recommends the use of filters on edge interfaces that specify the permitted IP ranges of source addresses. This is a fine strategy; router filters are well understand and lend themselves to this use. However, such filters are only appropriate at the network edge, as otherwise the number and complexity of IP ranges that would need to be specified in a filter would be prohibitive. Generally, the expected source addresses on network edge interfaces can be expressed with a small number of IP prefix statements.

Some motivated engineers have automated [11] the process of filtering addresses that should not appear anywhere on the Internet (''bogons''), for the purpose of using better access lists on core router interfaces. This strategy is a definite improvement, but suffers many drawbacks. It is almost certainly incomplete, as it relies only upon the IANA records of allocated IP address space. Thus, source addresses from ranges not in active use will be permitted to pass. Additionally, these filters can be difficult to maintain due to the methods available for changing access lists on routers. Finally, if the filter is not kept up to date, it can quickly become stale and prevent valid traffic from passing. Some operators became painfully aware [12] of stale filters when the 69/8 IPv4 space was recently allocated, as they were assigned IP address space that could not contact some Internet-connected hosts.

Other proposed strategies for address verification call for the use of special address verification tables in routers. The SAVE [13] protocol, for example, calls for additional communication between routers to exchange prefix path information. This differs from current inter-domain routing, as BGP routers select the best path to each destination and forward only this path to neighbor routers [14]. While additional inter-router messages might help solve the problem, vendors have not readily implemented such protocols. There are concerns about the effectiveness of such protocols in the absence of widespread utilization, as well as the additional overhead in implementing, securing, and running another protocol.

A solution that does not rely on static filter configuration is desired. While proactive sites might

deploy filters, for maximum penetration of SAV it is clear that there must be automatic methods of enabling it. Automatic verification mechanisms, however, still require a policy source to identify permitted source addresses. Ideally, this policy source must be dynamic, automatically reacting to network changes and blocking only illegitimate traffic. Finally, it must be nearly effortless on the part of most network operators to implement.

Unicast Reverse Path Forwarding

One method, unicast reverse path forwarding strict mode (uRPF-SM), promises to reduce administrative overhead and make source address verification feasible at more places of the network. Borrowing from the extensive use of reverse path forwarding checks in multicast routing, uRPF acts as an additional ingress filter on router interfaces. Specifically, routers will look up the source address of inbound packets in the unicast forwarding information base (FIB) to identify the next hop interface. If there is a route to the address via the same interface that the packet was received on, the packet is allowed to enter the router. Otherwise, the packet is rejected. This processing is shown below in Figure 6. Using uRPF, operators are free to define additional ingress filter lists, but no filter is required.

Because the mechanism uses the unicast forwarding base as the policy source, once administrators have setup a working network, address verification at the extreme edge is as simple as enabling uRPF on each interface. As network routes change, the filter is kept up to date without any operator involvement. However, it is important that correct forwarding entries exist to all potential source addresses on a particular interface. The important distinction is that of valid routes to a host versus the best route to the host. Only the best route(s) are installed into the forwarding table, which uRPF uses as its policy source. On intradomain networks, there is typically no cause for concern in this regard. However, edge interfaces connecting dually-homed machines may need to be carefully considered. Fortunately, some implementations provide an unconditional acceptance filter (uRPF ACL). That is, packets that would be otherwise rejected but match the configured filter will be forwarded.

While uRPF-SM is a good solution for address verification on the network edge, it is still unsuitable for use in the core. Due to network engineering requirements, packets may use asymmetric paths to travel between two hosts. In this case, since the forwarding information base contains only best-path routes, uRPF would not verify the source address and reject the packet. Because of a strong desire to provide even minimal SAV on internal nodes, an alternative to strict mode uRPF was introduced. Known as loose-mode uRPF (uRPF-LM), it requires only that the route to a particular network exist in the forwarding base. It does not require that the ingress interface match the next hop interface identified by the forwarding table entry. This provides a good alternative to the use of interface-based bogon access lists, as loose-mode uRPF will perform similar verification without requiring updates to the access lists (especially if bogon route filters are in place.)

Configuring

The basic steps for configuring unicast reverse path forwarding on a router are below. Specific configuration of these features on two router platforms is listed in Table 2.

- Ensure that IP unicast routing is properly configured and stable: For each router interface, identify all valid source addresses from the connected network and verify that routes to the interface exist in the forwarding table. This is primarily a concern on edge interfaces.
- Configure access lists for unconditional acceptance of specific source addresses as necessary.
- On edge router interfaces, specify uRPF strict mode.
- On core interfaces, specify uRPF loose mode.

Summary

Unicast reverse path forwarding provides an easy and self-maintaining mechanism for source address verification. Verification is a small but important piece in protecting the Internet infrastructure. In non-transit edge networks (most enterprises, for example), implementing uRPF-SM on edge interfaces should become standard operating procedure. On core interfaces in both transit and non-transit networks, uRPF-LM

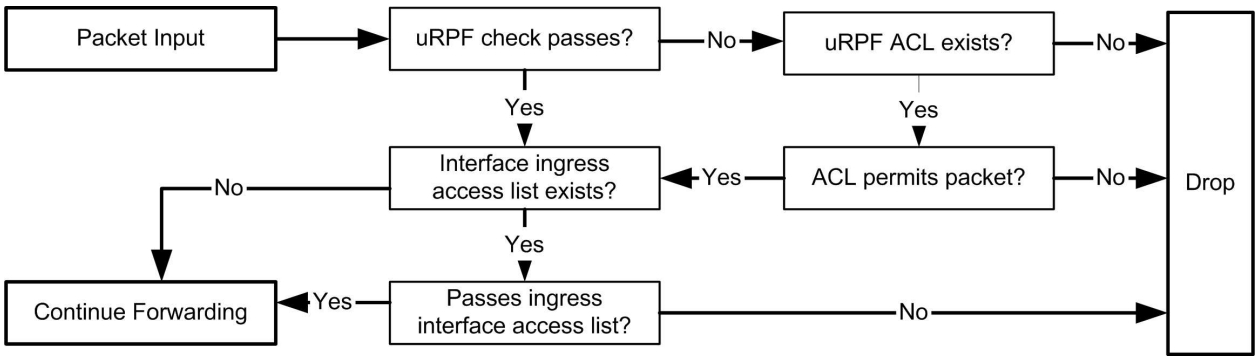

Figure 6: Unicast reverse path forwarding: Input packet filtering.

provides minimal verification that packets have source addresses with a valid return route. Implementing source address verification on edge interfaces of transit networks is potentially difficult, given multi-homing and traffic balancing of downstream connectors. For this reason, it is all the more important that sites able to deploy strict mode verification do so.

Using uRPF for Address Filtering

As network operators, we occasionally need to apply filters blocking certain source addresses from entering a router interface. In our network, there are two primary reasons for needing to do this. First, we sometimes want to block certain source addresses from entering at our border. This occurs when we are undergoing a network attack or otherwise have an administrative need to block packets from particular addresses. Additionally, we periodically need to prevent internal addresses from accessing the network. These blocks are typically applied to enforce administrative restrictions. For example, when a host is infected with a worm, we work quickly to filter the host, preventing further network abuse.

While either source or destination-based address filters can be applied using access list style restrictions on interfaces, changing these filters is time consuming and error prone. In the midst of network abuse, such as a high-rate denial of service attack, changing filters can be especially troublesome. Administrators might have difficult connecting to the router and, once connected, might face difficulties with low resource availability. Even in otherwise stable conditions, changing access filters can be a complicated process.

Destination Address Filters

As we look beyond the use of access filters, we note that it's relatively easy to apply destination address filters; that is, block traffic bound for specific addresses or subnets. Null routes can be added to a sinkhole router in a network, and with proper redistribution any traffic destined for the address will be drawn to the sink hole. Using null routes is typically easier than filters for destination-based blocking, especially if a host-based routing daemon is the null route source. This does mean that traffic is not blocked at the edge; instead it is dropped at the originating router. Because routers are optimized to forward traffic, however, this is typically an acceptable tradeoff.

Some interesting work has been done by operators seeking an efficient way to identify ingress access points of denial-of-service traffic. Working even in the face of spoofed source addresses, the Remote Triggered Blackhole Filtering [15] (RTBF) mechanism causes traffic destined for a specific address or subnet to be blocked at the network edge. Rather than adding a null route for the targeted address, instead an iBGP route to a specially-designated address is added. Each edge router is then configured with a null route to this special address. Packets to the blocked addresses are dropped at the network edge. By logging ICMP Unreachable message generation, operators can quickly locate the traffic ingress access point.

Source Address Filtering with uRPF

Unlike destination address filters, blocking packets from specific source addresses is harder to improve from traditional access lists. However, using uRPF-SM, we can enable faster filtering of specific source addresses. Using a host-based routing daemon, addresses or subnets to be blocked are configured as null routes and are redistributed through the routing domain. The strategies of RTBF to effectuate null routes at each router could also be used. Note that in most cases, we are talking about adding host routes to individual addresses to block.

Adding these null routes serves two purposes: any traffic destined for the target IP will be redirected to the announcing router. Additionally, while ACL-filtered source addresses would ordinarily pass the uRPF check (and then be dropped due to the ACL), announcing a more specific route (in most cases, a host route) will cause routers within the domain to have a more specific forwarding table entry for the filtered IP range. This specific entry will be in the direction of the announcing router, not the normal ingress interface. Thus, using uRPF-SM the traffic will be dropped. Figure 7 and Figure 8 illustrate the network changes when an address is black-holed by the host-based router.

This method is an improvement over the use of access lists, due to the relative ease with which source address filters can be effectuated. Host-based routing daemons provide the capability of easily scripting changes to the routing configuration. During network denial-of-service attacks, minimizing the time to apply filters translates into a faster recovery time. Additionally, this can relieve the burden on network operators to change router configurations, instead making the process of applying filters one that is more easily delegated.

The procedures to enable the use of uRPF for source address filtering are:

- Configure edge interfaces for uRPF-SM as described above.
- Set up a host-based router and configure it to be a part of the IGP routing infrastructure. In our case, this means enabling OSPF, specifying the

Platform	Strict Mode	Loose Mode
Cisco IOS	ip verify unicast source reachable rx	ip verify unicast source reachable any
Juniper JunOS	unicast-reverse-path active-paths	unicast-reverse-path feasible-paths

Table 2: uRPF configuration commands.

correct area parameters, and entering the proper authentication keys.

- Configure a mechanism for dynamically changing routes on the host router. We are developing a system to provide approved administrators with a mechanism for specifying hosts to filter or unfilter. Each host is specified by IP address.

When a host is filtered, the following occurs:

- A static host route (/32 route) is added to the routing daemon. This can occur manually or via an automated mechanism.
- The route is redistributed through the IGP. Routers hearing the announcement will install a host route in the direction of the announcing router.
- Traffic to the filtered host will be delivered to the host router, where it will be discarded.
- Traffic from the filtered host will be blocked by uRPF-SM on the first hop router. The forwarding table lookup will result in the next hop interface being towards the announcing host router, not towards the edge network. Thus, the traffic is dropped.

Deployment Example

Background

At Carnegie Mellon, we began implementing ingress filtering on some interfaces in 1998. These filters were applied to edge interfaces and specifically allowed the subnet(s) present on the interface. We expanded the filtering to every router edge interface in 2000. However, changing the filters by hand is an arduous process. As subnets are resized and moved, administrators must be vigilant in keeping the filters up to date.

Coincident with growing incidents of viruses, worms, and backdoors, we saw a rise in the number of times we needed to apply IP filters against particular

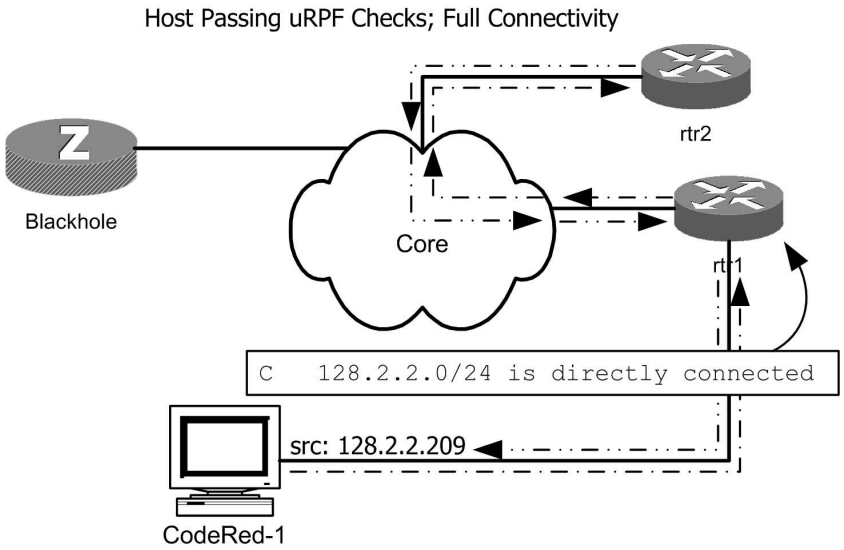

Figure 7: Host passing uRPF checks.

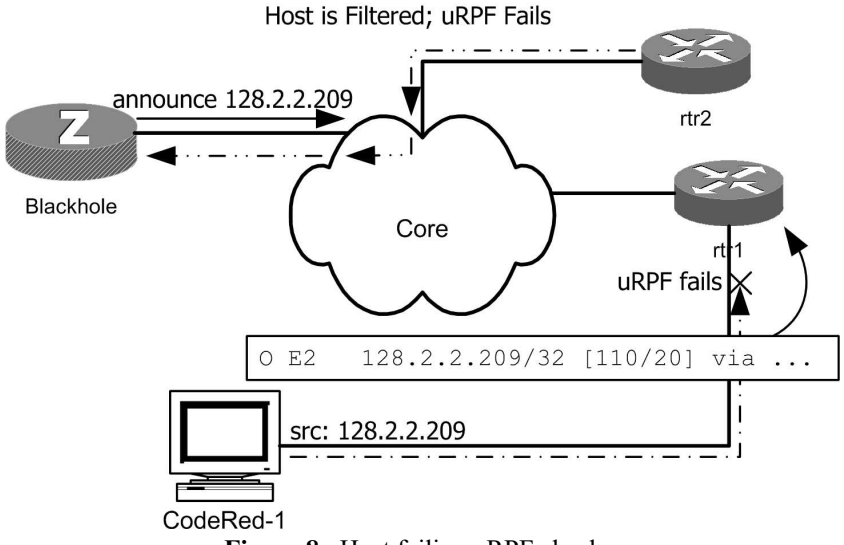

Figure 8: Host failing uRPF checks.

campus source addresses. Additionally, we regularly need to apply IP filters to enforce administrative restrictions. During the height of Code Red [16], for example, up to 40 filters per day were being added as machines were compromised. These filters were done as head-of-the-line additions to the filter on the ingress interface. Changing the filters manually is very time consuming and error-prone, and several cases of incorrect filters in our network were identified.

Needing to add and remove source filters frequently, we wrote a utility to automate the process of adding and removing the filters on edge interfaces. The initial version filtered a single host at a time, requiring administrators to enter passwords when necessary. Subsequent revisions can change multiple filters in one session to the router. The script intelligently uses the routing table to identify the ingress interface, identifies the current interface ACL, and then safely inserts or removes the requested elements.

In summary, the use of ingress filters at Carnegie Mellon serves two primary purposes: source address verification, and applying administrative restrictions through source address filtering. We can easily use uRPF-SM to implement source address verification without using interface ACLs. Using the methods described here for applying source address filters with uRPF, we nearly eliminate the need for access list filters in our network. Our IGP is OSPF and, as noted, we have experience using the Quagga OSPF routing daemon on Linux hosts to join our OSPF routing domain.

After uRPF-SM configuration on edge interfaces of our routers, we install a machine running the Quagga host routing daemon. This machine has connections to our network core, to minimize the number of routers processing traffic null routed on the host. When addresses need to be filtered, they are added as null routes to the Quagga configuration. Redistributed into our OSPF routing domain, backbone routers quickly install the host route into their forwarding table. uRPF-SM then blocks incoming traffic from the address, while traffic to the address is routed to the Quagga host. Note that the announcement is not propagated out of our IGP; this does not change our BGP announcements. Additionally, the OSPF announcement is subject to traditional OSPF inter-area flooding rules. For instance, routers in totally stubby areas will not hear the host route announcement. This procedure has come to be known as OSPF-based Remote Triggered Blackhole Filtering (oRTBF).

Summary

Once unicast reverse path forwarding is implemented in a network, it can be effectively leveraged to quickly apply source address filters. There are several advantages to applying filters in this manner beyond the obvious time savings. Operators need not locate where the address enters the network, saving additional time in applying filters. Finally, using a host-based routing daemon makes it easier to delegate administrative control of filter operations. Operators may even consider using more than one host to duplicate the null route announcements, leading to greater reliability and stability of the address filtering.

Conclusion

Anycast addressing, host-based router daemons, and unicast reverse path forwarding are three useful tools in the ever-expanding network operator's toolbox. Our experiences show that as applied to the problems addressed here, they can provide solutions that improve network reliability and save operators' time. The need for source address verification is well understood by most backbone network operators, but the use of verification in enterprises remains low. Where possible, operators should investigate enabling uRPF on all edge network interfaces, as this directly enhances network security.

Author Information

Kevin C. Miller is a Network Systems Developer at Carnegie Mellon, where he develops management and monitoring tools to support the campus network. He graduated from Carnegie Mellon with a BS in CS, and can be reached at <kcm@cmu.edu>. Additional resources related to this paper are available from http://www.net.cmu.edu/pres/lisa03 .

References

[1] Partridge, C., T. Mendez, and W. Milliken, "Host Anycasting Service," *RFC1546,* BBN, http://www.ietf.org/rfc/rfc1546.txt , November, 1993.

[2] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," *RFC2893*, FreeGate Corp., http://www.ietf.org/rfc/rfc2893.txt .

[3] Internet Software Consortium, "F.root-servers.net," http://www.isc.org/services/public/F-root-server.html .

[4] "ISC BIND 9," http://www.isc.org/products/BIND/bind9.html .

[5] Moy, J., "OSPF Version 2," *STD54, RFC2328*, Ascend Communications, http://www.ietf.org/rfc/rfc2328.txt , April, 1998.

[6] "Quagga Routing Suite," http://www.quagga.net .

[7] Murphy, P., "The OSPF Not-So-Stubby Area (NSSA) Option," *RFC3101*, U.S. Geological Survey, http://www.ietf.org/rfc/rfc3101.txt , January, 2003.

[8] "Advisory CA-1998-01; Smurf IP Denial-of-Service Attacks," CERT, http://www.cert.org/advisories/CA-1998-01.html .

[9] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," *BCP38, RFC2827*, http://www.ietf.org/rfc/rfc2827.txt , May, 2000.

[10] "Advisory CA-1997-28; Denial-of-Service Attacks," CERT, http://www.cert.org/advisories/CA-1997-28.html.

[11] "Bogon List," http://www.cymru.com/Documents/bogon-list.html.

[12] Mauch, J., "The 69.0.0.0/8 Problem," http://puck.nether.net/~jared/papers/69-paper.html, April, 2003.

[13] Li, J., J. Mirkovic, M. Wang, P. Reiher, and L. Zhang, "SAVE: Source Address Validity Enforcement Protocol," *Proceedings of the IEEE Infocom*, 2002.

[14] Varadhan, K., R. Govindan, and D. Estrin, "Persistent Route Oscillations in Inter-domain Routing," Tech. Rep. 96-631, USC/ISI, February, 1996.

[15] Greene, B. R., "Remote Triggering Black Hole Filtering," Cisco Systems, http://www.cisco.com/public/cons/isp/essentials/Remote_Triggered_Black_Hole_Filtering-02.pdf, August 2002.

[16] "Advisory CA-2001-19: 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL," CERT, http://www.cert.org/advisories/CA-2001-19.html.