USENIX Association

Proceedings of the 2001 USENIX Annual Technical Conference

Boston, Massachusetts, USA
June 25–30, 2001

A Waypoint Service Approach to Connect Heterogeneous Internet Address Spaces

T. S. Eugene Ng, Ion Stoica, Hui Zhang
Carnegie Mellon University
Pittsburgh, PA 15213
{*eugeneng, istoica, hzhang*}*@cs.cmu.edu*

Abstract

The rapid growth of the Internet has made IP addresses a scarce resource. To get around this problem, today and in the foreseeable future, networks will be deployed with reusable-IP addresses (a.k.a. private-IP addresses) or IPv6 addresses. The Internet is therefore evolving into a collection of networks of heterogeneous address spaces. Such development jeopardizes the fundamental bi-directional connectivity property of the Internet.

The problem is that, without IP addresses, non-IP hosts (i.e. reusable-IP or IPv6 hosts) cannot be directly addressed by IP hosts, making it impossible for IP hosts to initiate connections to them. To solve this problem, we propose a network layer waypoint (3rd-party network agent) service called AVES. The key idea is to *virtualize* non-IP hosts by a set of IP addresses assigned to waypoints. The waypoints then act as relays to connect IP hosts to non-IP hosts. This scheme allows every IP host to simultaneously connect to as many non-IP hosts as the number of waypoint IP addresses. Therefore high connectivity is achieved by AVES even when a small number of IP addresses are used. Unlike other known solutions, AVES can provide general connectivity and does not require any change to existing IP hosts or IP network routers for easy deployment. We have implemented and deployed an AVES prototype system at CMU. A wide range of applications have been shown to work seamlessly with AVES. Details of our implementation's design, performance and limitations are discussed.

1 Introduction

The Internet was originally conceived as a homogeneous global network in which all hosts would implement the network protocol Internet Protocol version 4 (IP or IPv4) [20], have globally unique unicast IP addresses for identification and routing purposes, and could freely communicate with each other. But as the Internet evolves, it is becoming a heterogeneous network (as depicted in Figure 1(a)). In the process, *bi-directional* connectivity between hosts is lost. That is, given a pair of hosts, sometimes a connection can be established only if it is initiated by a particular side, and sometimes a connection cannot be established at all.

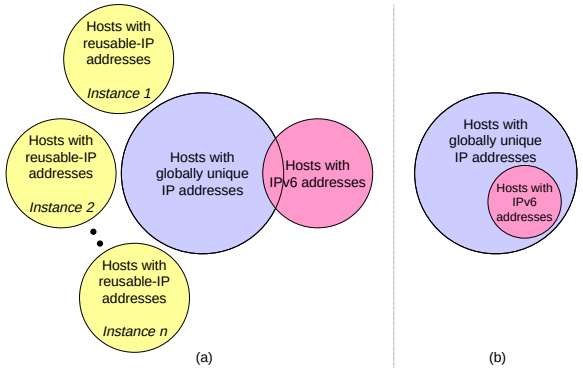

Figure 1: (a) Heterogeneous address spaces (b) IPv6 dual-stack strategy

The root of the problem is that with the rapid growth of the Internet and the inefficient utilization of the IP address space, it has become clear that the relatively small 32-bit address space defined by IP is insufficient. The danger of exhausting the IP address space has prompted the Internet Assigned Numbers Authority (IANA) to conserve the remaining IP address space. This has resulted in two important development trends.

First, to get around the IP address shortage problem, it is increasingly common for networks ranging from large corporate networks to small home networks to be deployed using reusable-IP addresses.[1] By connecting a reusable-

This research was sponsored by DARPA under contract number F30602-99-1-0518, and by NSF under grant numbers Career Award NCR-9624979, ANI-9730105, ITR Award ANI-0085920, and ANI-9814929. Additional support was provided by Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

[1] Reusable-IP addresses are the network prefixes 10/8, 172.16/12 and 192.168/16 [21]. We use the term "reusable-IP addresses" instead of the more conventional "private-IP addresses" to distinguish from another use of these addresses to build *private* IP networks that are intentionally made inaccessible to the public Internet.

IP network to the IP Internet through a Network Address Translation (NAT) [25] gateway, *uni-directional* connectivity to the IP Internet is provided. That is, in general, reusable-IP hosts can initiate connections to IP hosts but not vice versa. Moreover, between two reusable-IP hosts belonging to different networks, there is generally no connectivity. Thus, hosts inside reusable-IP networks are not first-class Internet entities.

Second, as a long term solution, the IETF has designed the Internet Protocol version 6 (IPv6) [4] which defines an enormous 128-bit address space. Ideally, all new networks should now be deployed using IPv6, and all existing IP and reusable-IP networks should be upgraded to IPv6. However, since upgrading to IPv6 is a gradual process, IP and reusable-IP networks will remain in the foreseeable future. In addition, although new IPv6 networks can be fully compatible with IP when the dual-stack transition mechanism [8] is used, to achieve *full transparency*, every IPv6 host must be assigned an IP address and essentially behave as both an IPv6 and an IP host simultaneously as shown in Figure 1(b). Obviously, for many IPv6 network operators, this is simply not a viable option. Thus, a significant portion of IPv6 networks will likely be deployed as IPv6-*only* networks, and they will only have *uni-directional* connectivity to the IP Internet via Network Address Translation - Protocol Translation (NAT-PT) [27] gateways similar to the reusable-IP network scenario.

These development trends clearly indicate that the Internet today and in the foreseeable future will be a heterogeneous network composed of IP, IPv6 and reusable-IP address spaces as shown in Figure 1(a), and the fundamental bi-directional connectivity property of the Internet has been destroyed. In this environment, many common applications are no longer usable. Recent interest in peer-to-peer applications has raised awareness of this problem because under these applications there is no longer a distinction between client versus server and bi-directional connectivity is crucial. An important challenge is: *How can the lost connectivity in this heterogeneous environment be restored to as high a degree as possible*? The obvious difficulty is that, without IP addresses, non-IP hosts (i.e. reusable-IP or IPv6 hosts) cannot be directly addressed by IP hosts, therefore IP hosts cannot initiate connections to non-IP hosts directly. Any general solution to this problem must therefore allow a non-IP host to be identified by an identifier other than an IP address, and the identifier must be mapped to the actual non-IP host during communication.

To date, no known solution to this problem can provide *general* bi-directional connectivity and at the same time be deployed easily. Of the known solutions, which are discussed in Section 7, some are specific to one application (e.g. HTTP virtual hosting), some are application independent but require per application manual configurations and cannot provide general bi-directional connectivity (e.g. port forwarding), and some can provide general bi-directional connectivity but require upgrades to existing IP hosts or IP network edge routers (e.g. SOCKS-based proposal). In practice, these upgrades to existing IP hosts or IP network edge routers are either too daunting to carry out, or there is no incentive to carry them out in the first place because they are aimed to benefit non-IP hosts and do not directly benefit existing IP hosts and networks.

In this paper, our aim is to design a solution that not only provides *general* bi-directional connectivity but also requires as little upgrades to existing software and hardware as possible. To achieve this goal, we propose a network layer waypoint service called AVES. Waypoints are 3rd-party network agents. The key idea is to *virtualize* non-IP hosts by a set of IP addresses assigned to waypoints. In this approach, we use DNS [15] names as identifiers for non-IP hosts and *dynamically* bind non-IP hosts to waypoint IP addresses during DNS name resolution in a *connection-initiator-specific* fashion. The waypoints then act as relays to connect IP hosts to non-IP hosts through AVES-aware NAT gateways.[2] This scheme allows every IP host to simultaneously connect to as many non-IP hosts as the number of waypoint IP addresses. As a result, high connectivity is achieved even when a small number of IP addresses are used. The internetworking heterogeneity is handled by the waypoints, no upgrade to existing IP hosts or IP network routers is required, making non-intrusive deployment of AVES possible. This approach is unique because it addresses an internetworking problem without changing the network layer of existing systems besides the NAT gateways.

It is important to note that AVES is optimized for deployment and is not perfect in every regard. In particular, AVES trades performance for deployability. It turns out that, since the binding of non-IP hosts to waypoint IP addresses during DNS name resolution is the critical step, the more control we have over the local DNS servers used by IP initiators, the better AVES can perform. However, in the extreme case where we have no control over the local DNS servers, AVES still provides the same connectivity but at the cost of lowered performance.

In Section 2, we further motivate the heterogeneous address space connectivity problem with a case study and precisely formulate the problem. We present the design of AVES in Section 3, and discuss its connectivity and deployability properties in Section 4. We have implemented a complete prototype of AVES on Linux and the details are presented in Section 5. In Section 6, we discuss key concerns about AVES such as application compatibility, scalability, and security. Finally, we discuss related work in Section 7 and summarize the paper in Section 8.

[2]Note that no known solution can provide general bi-directional connectivity without extending the functionality of the NAT gateway. However, since the operator of a NAT gateway has incentives to perform the upgrade, deployment should not be hindered.

		Responder		
		IP	IPv6	R-IP
Initiator	IP	Trivial	(a) Hard	(b) Hard
	IPv6	NAT-PT	Trivial	Reduces to (b)
	R-IP	NAT	Reduces to (a)	Reduces to (b)

Table 1: Taxonomy of address space connectivity

2 Case Study and Problem Formulation

To further motivate the need for bi-directional connectivity across heterogeneous address spaces, let us consider the DSL service at CMU. In April 1999, CMU began offering an internal DSL service that allowed users to obtain as many IP addresses as needed. Twenty months later, the 2000 IP addresses allocated to the service were exhausted. To conserve IP addresses, today only one statically assigned and one dynamically assigned IP address is provided per DSL line.

The situation has driven many of our DSL users to begin using NAT to get around the address allocation problem. Unfortunately with NAT, bi-directional connectivity is lost. This drastically affects the user's computing activities because fundamentally the university environment is not a pure client-server environment and bi-directional connectivity is critical. Although the DSL user will still be able to browse the web from home and access campus computing resources, she will not be able to remote login directly to her home computers using `ssh` or `telnet`. She also will not be able to host her own web servers or `ftp` servers on her home computers to distribute documents like digital videos and photos. When she is accessing campus computing resources from home, she also will not be able to bring up `X Windows` applications on her home computers (unless `ssh X Windows` connection forwarding is used). Many popular peer-to-peer applications also break down. For example, when both parties are behind NAT gateways, the popular music sharing software Napster will not work.

In Section 7, we discuss a simple port number forwarding work-around that can partially address these problems. However, this work-around works in the transport layer, requires per application manual configurations, and the connectivity achieved is unacceptable as only one home computer per port number can accept in-bound connections. In contrast, as we shall see, AVES is capable of allowing DSL users who deployed NAT gateways to fully regain all the above lost capabilities.

2.1 Heterogeneous Address Space Connectivity Problem

In the foreseeable future, three types of address spaces will coexist in the Internet, they are IP, IPv6, and reusable-IP. Table 1 describes the connectivity between all combinations of the three address space types. In a connection, the initiator is the "caller" host that sends the first packet to start the connection; the responder is the "callee" host that answers the in-coming connection. For example, to connect a reusable-IP (R-IP) initiator to an IP responder, it is well known that NAT [25] can be used and it works well in practice. Similarly, NAT-PT [27] can be used to connect an IPv6 initiator to an IP responder. On the other hand, to connect an IP initiator to an IPv6 responder (case (a), Table 1), or to connect an IP initiator to a reusable-IP responder (case (b), Table 1) is hard because the responder does not have any IP address and the initiator cannot address the responder directly. Solving these problems is the key challenge in maintaining the bi-directional connectivity abstraction of the Internet.

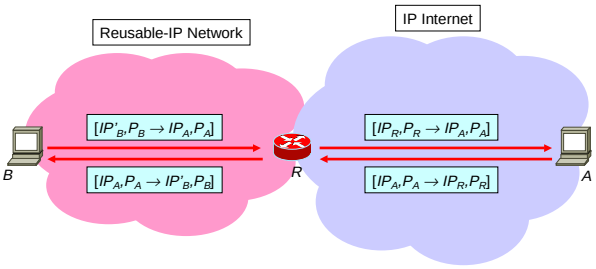

Figure 2: Out-bound connectivity via NAT gateway

We emphasize that the problems underlying case (a) and case (b) are essentially identical, except that case (a) requires additional packet header format conversion which has been well documented in [18, 27]. Thus for simplicity, for the remainder of this paper, we only consider case (b), where an IP initiator is connecting to a reusable-IP responder. The results can be mapped to case (a).

Note that because there are multiple coexisting instances of the reusable-IP address space, connecting a reusable-IP initiator to a reusable-IP responder in a different instance of the address space is non-trivial. However, under NAT, this is equivalent to the initiator's NAT gateway (which is an IP host) connecting to the reusable-IP responder. Therefore, this case can be reduced to case (b) as indicated in Table 1. Similarly, connecting a reusable-IP initiator to an IPv6 responder reduces to case (a), and connecting an IPv6 initiator to a reusable-IP responder reduces to case (b).

In summary, the key difficulty in achieving bi-directional connectivity across heterogeneous address spaces is to provide connectivity from IP hosts to non-IP hosts.

2.1.1 NAT and Its Limitation

It is helpful to understand the capability and limitation of NAT, but as we shall see, NAT can only provide uni-directional connectivity to the IP Internet. Figure 2 illustrates a typical scenario where a network is constructed using the reusable-IP address space and is attached to the IP Internet via a NAT gateway, *R*.

Assume *R* only owns a single IP address. Consider the

case where a reusable-IP host B (the initiator) is connecting to an IP host A (the responder). A reusable-IP address that belongs to host X is denoted IP^{l}_{X}, and an IP address that belongs to host Y is denoted IP_Y. Assume B already knows the IP address of A.[3] B simply initiates the connection by sending a packet to A. Suppose this is a TCP connection, and the packet sent by B has a source port number P_B and a destination port number P_A. We denote this packet by $[IP^{l}_{B}, P_B \rightarrow IP_A, P_A]$ (the transport protocol is omitted for simplicity). The goal of NAT is to represent B in the IP Internet by R. As this packet is forwarded by R, R replaces IP^{l}_{B} by its own IP address IP_R, and P_B by an available port number on R, say, P_R. The resulting packet is $[IP_R, P_R \rightarrow IP_A, P_A]$ and is forwarded out of the reusable-IP network. When a corresponding response packet $[IP_A, P_A \rightarrow IP_R, P_R]$ is received by R, R simply replaces the destination address by IP^{l}_{B} and the destination port number by P_B. Since each 16-bit port number on R can be reused for different transport protocols, roughly 65,000 TCP and 65,000 UDP connections can be simultaneously active from initiating reusable-IP hosts to every port of every responding IP host even though R only has one IP address.

In contrast, if A is the initiator and B is the responder, the situation becomes very different. Because the only IP address owned by the reusable-IP network is IP_R, a DNS application level gateway [26] for in-bound NAT must resolve the name lookup for B to IP_R. Unfortunately, since IP_R can only refer to one reusable-IP host at any given time, with one IP address, NAT can only provide general in-bound connectivity to one responder in the entire reusable-IP network at a time. Since having one IP address is typical, NAT cannot provide acceptable in-bound connectivity.

3 AVES

In this section, we describe AVES (Address Virtualization Enabling Service), which can non-intrusively provide IP to IPv6 or IP to reusable-IP connectivity. Again, for simplicity, we only consider the reusable-IP scenario. The discussion also applies to the IPv6 scenario. For non-IP to IP connectivity, we simply rely on NAT and NAT-PT.

3.1 Overview

The key idea behind AVES is to *virtualize* non-IP hosts by a set of IP addresses assigned to waypoints. The waypoints then act as relays to connect IP hosts to non-IP hosts. Figure 3 illustrates this idea. In this example, there are two reusable-IP networks connected to the IP Internet, and the reusable-IP hosts B and C are virtualized by the IP addresses of waypoints W_2 and W_4. As a result, IP initiators A and D can connect to responders B and C through the

[3]Such an IP address is usually obtained through DNS, and since a DNS server's address is known by configuration, we can assume any IP host's address can be known by B without any loss of generality.

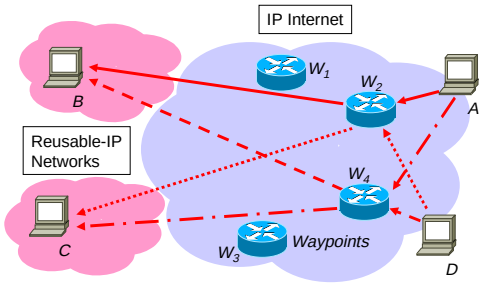

Figure 3: Overview of AVES

waypoints. Note that the bindings between non-IP hosts and waypoint IP addresses are *initiator-specific*. That is, each IP initiator has its own view. In our example, to A, B is bound to IP_{W_2} and C is bound to IP_{W_4}. On the other hand, to D, B is bound to IP_{W_4} and C is bound to IP_{W_2}. This ability to simultaneously bind an unlimited number of non-IP hosts to a waypoint IP address allows AVES to provide connectivity to an unlimited number of non-IP hosts. Another point worth noting is that the number of waypoint IP addresses only limits the number of non-IP hosts that each IP initiator can simultaneously connect to. Thus, for all practical proposes AVES requires only a small number of IP addresses, say a few tens, to achieve high connectivity.

More precisely, to implement AVES, a service provider deploys a small number of IP waypoints (W_1 - W_4) and AVES-aware DNS servers (not shown) for the reusable-IP domains. The waypoints have the following characteristics: (1) Waypoints are assigned IP addresses, possibly more than one per waypoint, in which case each IP address is logically a distinct waypoint. Here we assume only one IP address is assigned per waypoint. (2) Waypoints are capable of performing address (and protocol, in the case of IPv6) translation, they serve as relays for traffic crossing heterogeneous address spaces. (3) Because waypoints are network agents, they can be deployed non-intrusively without global coordination. Under AVES, for IP initiator A to connect to reusable-IP responder B, it first performs a DNS name lookup for B; this marks the beginning of a *session*. The name lookup operation serves two purposes. First, the DNS name will uniquely identify the responder even though it does not have a unique IP address. Second, when the DNS query is processed by an AVES-aware DNS server, the non-IP host is bound to the IP address of a chosen waypoint, in this case IP_{W_2}. Again, this binding is initiator-specific so that a waypoint IP address can be bound to multiple non-IP hosts simultaneously. Instructions are then sent by the AVES-aware DNS server to W_2 so that it can correctly relay packets. IP_{W_2} is returned to A in the DNS reply with the time-to-live field set to zero (i.e. no caching of IP address records is allowed; however name server records can be cached). The session is now established, and A can open arbitrary connections to B through W_2. A session is terminated when a timeout occurs after

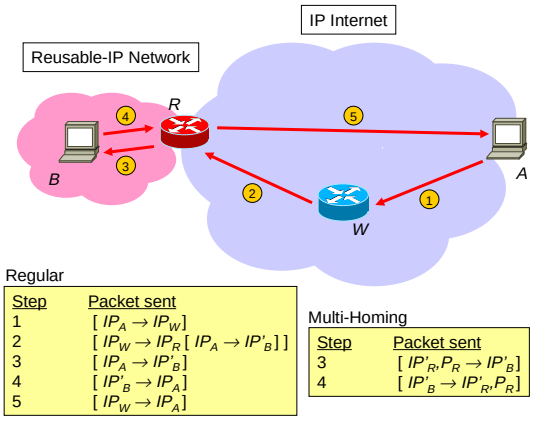

Figure 4: Data path operations

a period of inactivity. Afterwards, an initiator can regain connectivity by starting another session. This example illustrates several key ideas underlying AVES:

- **Virtual expansion of IP address space** – A waypoint IP address can virtually represent, or virtualize, an unlimited number of non-IP responders in the IP Internet simultaneously because the binding is *initiator-specific*. Hence, AVES virtually expands the IP address space, achieving high connectivity when only a small number of IP addresses are used.
- **Heterogeneity hiding** – From the point of view of an IP initiator, with AVES, all responders appear to be IP hosts with IP addresses. Thus, there is no need to modify existing IP hosts or IP network routers to achieve connectivity.
- **Transparent access** – An initiating IP host accesses AVES transparently via DNS host name resolution. The IP address of the selected waypoint is returned to the initiating IP host. The service abstraction provided by AVES is therefore simply an IP address, which is most compliant with existing applications.

In the following, we first explain the data path operations, then we explain the control path operations for configuring the data path and discuss deployment scenarios in relation to our case study in Section 2. The connectivity achieved by AVES is summarized precisely in Section 4.

3.2 Data Path Operations

Figure 4 shows a typical data path between an initiator A and a reusable-IP responder B. W is a waypoint and R is an AVES-aware NAT gateway. W virtualizes B for A. Thus, to A, the IP address of B is IP_W. To correctly relay packets from A to B, W has been configured by an AVES-aware DNS server via the control path protocol described in Section 3.3 with the following translation table entry (we omit the port numbers as they are unimportant):

Original packet	Translated packet	Encapsulation header
$[IP_A \rightarrow IP_W]$	$[IP_A \rightarrow IP'_B]$	$[IP_W \rightarrow IP_R]$

That is, when a packet from IP_A is received by W (recall that the binding is initiator-specific), the destination address of the packet is translated to IP'_B, and the resulting packet is tunneled from IP_W to IP_R. Note that we propose a tunneling based mechanism here despite the header overhead because the encapsulation header allows complete information about the session to be carried along with each data packet so that R can process each in-coming data packet purely based on its packet headers. This eliminates the need for a control path mechanism to configure R ahead of time, resulting in a simpler protocol. In the following, we describe two versions of the data path operations. The first version applies when the reusable-IP network is connected to the IP Internet via a single NAT gateway. The second one applies when the reusable-IP network is "multi-homed", that is, it is connected to the IP Internet via multiple NAT gateways.

The data path operations without multi-homing support are as follows. A initiates a connection to B by sending the packet $[IP_A \rightarrow IP_W]$ (step 1). When W receives such a packet, it transforms the packet into $[IP_A \rightarrow IP'_B]$ and encapsulates the packet with the header $[IP_W \rightarrow IP_R]$. We denote the final packet by $[IP_W \rightarrow IP_R[IP_A \rightarrow IP'_B]]$. To enhance security, this packet is authenticated by W. The packet is then forwarded (step 2) and later received by R. In addition to supporting the basic functionalities of a NAT gateway, R is extended such that when R receives an authentic encapsulated packet from W, it first determines whether a packet of the same connection (matching addresses in both outer and inner packet headers and port numbers, if any) has been seen before. If not, R creates a local translation table entry such that, when a corresponding out-bound packet $[IP'_B \rightarrow IP_A]$ (with matching port numbers, if any) is received, it will modify this out-bound packet to $[IP_W \rightarrow IP_A]$ before forwarding it out of the reusable-IP network. After creating this translation table entry, R removes the encapsulating packet header from the in-coming packet and forwards the inner packet to B (step 3). Finally, when B sends a reply to A (step 4), the packet $[IP'_B \rightarrow IP_A]$ is modified by R to $[IP_W \rightarrow IP_A]$ and then forwarded to A (step 5). Through these mechanisms, a connection from A to B is established.

The operations above prevent a reusable-IP network from being multi-homed because they do not guarantee that the out-bound packets of a session will traverse the same NAT gateway as the in-bound packets, consequently out-bound packets might not be translated correctly. To accommodate a multi-homed network, we modify the data path operations as follows. In step 3, the source address of an in-bound packet is translated to the reusable-IP address of R (IP'_R), and the source port number is translated to a chosen number (P_R) to maintain the binding. The resulting packet for step 3 is $[IP'_R, P_R \rightarrow IP'_B]$, and the packet for step 4 is $[IP'_B \rightarrow IP'_R, P_R]$. As a result, out-bound packets are guaran-

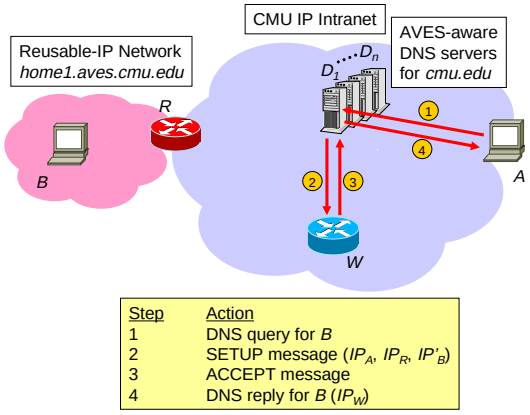

Figure 5: Control path operations for intranet deployment

teed to traverse the same border gateway as in-bound packets. For an in-bound ICMP [19] query packet, the Identifier field can be translated instead since there is no port number.

We have implemented both variations of the data path in our prototype system, see Section 5 for details. Limitations of these mechanisms are discussed in Section 6.

3.3 Control Path Operations

The AVES control path mechanisms are used to configure the data path. That is, when a DNS query for a reusable-IP responder is processed by an AVES-aware DNS server, a waypoint is selected to virtualize the reusable-IP responder and the appropriate translation table entry is installed at the selected waypoint so it can correctly relay packets.

It is important to recall that the bindings between waypoint IP addresses and reusable-IP responders must be *initiator-specific*. This allows a waypoint IP address to simultaneously virtualize many reusable-IP responders, and as a result high connectivity can be achieved with a small number of IP addresses assigned to waypoints. Unfortunately, creating initiator-specific bindings is not easy because the IP address of the initiator is typically not available in a DNS query received by an AVES-aware DNS server. This is because, in practice, virtually all end host systems implement *recursive* DNS query [15]. That is, an end host sends a *recursive* DNS query to its local DNS server, and this local DNS server generates additional *iterative* queries on behalf of the end host, and eventually returns the answer to the end host. Thus, an AVES-aware DNS server generally only interacts with the local DNS server of the initiator, the IP address of the initiator is obscured. In the following, we describe two deployment scenarios and the appropriate techniques in each case to create initiator-specific bindings.

3.3.1 Scenario 1 – Intranet Deployment

Let us reconsider the scenario discussed in Section 2. CMU can deploy AVES to restore bi-directional connectivity within the CMU intranet so that DSL users will be able to access their home computers directly from any host within the CMU intranet. To do so, CMU would deploy waypoints and upgrade its local DNS servers to make them AVES-aware. By upgrading the local DNS servers, initiator-specific bindings can be created easily since an initiator's IP address is now available in the IP headers of its DNS queries to the AVES-aware local DNS servers.

Figure 5 shows how this scheme works. Under this scheme, reusable-IP networks will use a common domain name suffix, say `aves.cmu.edu`, for easy identification. In our example, the reusable-IP network has a domain name `home1.aves.cmu.edu`. D_1 - D_n are upgraded AVES-aware local DNS servers. The control path operations are as follows. Initiator A's DNS query for B is directly sent to one of the AVES-aware local DNS servers, D_1 (step 1). D_1 is by configuration aware of the IP address of the AVES-aware NAT gateway R and the reusable-IP address of B. Upon receiving the DNS query, D_1 selects at random a waypoint among a set it knows, in this case W, and sends a SETUP message to W (step 2).[4] The SETUP message contains IP_A, IP_R, and IP'_B, which are necessary to create a data path translation table entry on W. When W receives the SETUP message, it examines its data path translation table to see if it can accept the request. Let us denote a translation table entry E on W more compactly by $\{IP_{initiator}, IP_{NAT}, IP'_{responder}\}$. Then, W can accept the request for initiator IP_A, NAT gateway IP_R, and responder IP'_B if and only if,

$$\forall E,\ IP_{initiator} = IP_A \Rightarrow (IP_{NAT}, IP'_{responder}) = (IP_R, IP'_B).$$

That is, if W already has a translation table entry for initiator IP_A, and the responder of that entry is not the same as the one in the SETUP message, then W must reject the request and reply with a REJECT message because W cannot be used to relay a particular initiator to more than one responder. On receiving a REJECT message, for simplicity, the AVES-aware DNS server will simply do nothing and let the initiator perform the DNS name lookup again to retry. In our example, the admission control criterion is satisfied, so W accepts the request, creates the corresponding translation table entry, and sends back an ACCEPT message (step 3). Finally, when D_1 receives the ACCEPT message, it responds to A's DNS query for B with the IP address of the selected waypoint, IP_W, with the time-to-live field set to zero (step 4). Note that the messages between waypoints and the AVES-aware DNS servers are authenticated to prevent unknown sources from gaining control of the system. Also, the messages can be lost in the network. Waypoint failure and packet loss are simply handled by initiator A's DNS query timeout/retry mechanism. Limitations of this scheme are discussed in Section 6.

[4] Selecting a waypoint based on performance metrics is a topic for future research.

```
X ← {};
on receiving new connection packet [IP_S → IP_W] :
    if IP_S should be rejected
        discard packet;
        return;
    if ∃E s.t. IP_initiator = IP_S and
            (IP_NAT, IP_responder) ≠ (IP_R, IP'_B)
        /* violation */
        discard packet;
        X ← X ∪ {IP_S};
    else
        accept packet;
        if ∄E s.t. IP_initiator = IP_S
            create E = {IP_S, IP_R, IP'_B};
on exiting wait state after T_wait :
    reject connections from ∀IP_S ∈ X for T_reject;
```

Figure 6: Waypoint wait state algorithm for general deployment

3.3.2 Scenario 2 – General Deployment

There are two major disadvantages of the previous deployment scheme. First, reusable-IP hosts are still unreachable from hosts that do not belong to CMU's intranet. Second, deployment requires upgrading CMU's local DNS servers and thus requires CMU's consent.

It is possible to overcome both of these short-comings by using a technique called *delayed binding* at the expense of lowered performance. The basic idea is that, a waypoint does not need to know the identity of the initiator to accept a request. It can accept the request optimistically and wait for the connection from the initiator to arrive, and only at that time admission control is performed and the actual binding is created.

Under this scheme, reusable-IP networks will use a common domain name suffix that is independent of any organization, say `avesnet.net`. Waypoints and AVES-aware DNS servers are independently deployed for the `avesnet.net` domain. No upgrade to any existing DNS server is needed. When a DNS query is received by an AVES-aware DNS server for `avesnet.net`, although the initiator's IP address (IP_A) is no longer known, the AVES-aware DNS server can still select a waypoint W and send it a SETUP message containing IP_R, IP'_B, and IP_{DNS} (the IP address of the initiator's local DNS server). Without knowing IP_A, W can no longer perform the admission control test stated in Section 3.3.1. However, W can make use of whatever information it has and decide whether to accept the request (in the simplest case, W always accepts the request). If W accepts the request, it replies with an ACCEPT message, and immediately enters a *wait state* for a short period of time, T_{wait}, and executes the algorithm shown in Figure 6. During this time, W does not accept other in-coming SETUP requests. Thus, requests are serialized.

In summary, during this wait state, when a new connection from some initiator S arrives (indicated by a TCP SYN packet or any non-TCP packet), S is potentially the initiator that W is waiting for. Thus, W checks to see if S violates the admission control criterion (note that E in Figure 6 denotes a waypoint translation table entry as defined in Section 3.3.1). If so, the packet must be rejected, and S is recorded in the set X of violators. If later a new connection from initiator A arrives, and A does not violate the admission control criterion, and W has no existing translation table entry for A, then a new translation table entry is created for A and bound to responder B. Upon exiting the wait state, connections from initiators in X must be rejected for a time period T_{reject} to force these initiators to retry their connections. Note that T_{reject} should not be too large or it may negatively affect future requests from the same initiator.

We have fully implemented delayed binding in our prototype system and it works well (see Section 5 for details). Since this technique is independent of organizational boundaries, it is actually feasible for our prototype system to provide service to reusable-IP networks outside of CMU.

One disadvantage of delayed binding is that connections need to be retried whenever an admission control violation is committed. Fortunately, when the number of waypoints is greater than the average number of simultaneous sessions opened by an initiator, the chance of this can be kept small. Another disadvantage is that the peak rate at which the whole system can accept new sessions is limited to N/T_{wait} sessions per second, where N is the number of IP addresses assigned to waypoints. Our prototype system, with 50 IP addresses and a T_{wait} of 2 seconds, can accept 25 sessions per second. While this is quite reasonable for CMU's DSL users, we do not advocate the use of our system to serve a popular web server. Other limitations regarding security and state consistency are discussed in Section 6.

3.3.3 Final Comment

Note that if we can extend the DNS protocol to always carry the original initiator's IP address in all DNS queries, deployment of AVES can be greatly simplified. General deployment can be achieved without making existing DNS servers AVES-aware or using delayed binding.

4 AVES Connectivity and Deployability

In Section 3.2, we described two data path designs. Assuming N IP addresses are assigned to waypoints, the in-bound connectivity achieved by each design is as follows. For the regular data path design without support for multi-homing:

- All hosts in non-IP networks are simultaneously reachable directly by IP hosts, regardless of the size of N.
- Each IP host can simultaneously open N *sessions* to reach a maximum of N non-IP hosts.

With multi-homing support, since port or identifier numbers are used for connection demultiplexing, the following additional restriction is imposed:

- Each port number of each non-IP host can be reached by no more than 65,000 TCP and 65,000 UDP connections simultaneously through each AVES-aware NAT gateway. Also, through each AVES-aware NAT gateway, each non-IP host can be reached by no more than 65,000 ICMP connections simultaneously. If a protocol does not use port or identifier number, then each non-IP host can only be reached by one connection of such protocol through each AVES-aware NAT gateway at a time.

Thus, as long as N is greater than the average number of simultaneous *sessions* to non-IP hosts opened by a typical IP initiator, say $N = 50$, in-bound connectivity can be restored to a high level.

To summarize AVES's deployability, waypoints can be independently deployed; NAT gateways need to be extended, however this is necessary and acceptable because their operators have the right incentives to perform the upgrade. To deploy AVES for an intranet, upgrading the local DNS server software will provide the best performance. However, even when it is impossible to upgrade existing DNS servers, the delayed binding technique can be used at the expense of lowered performance. In all cases, no existing IP hosts or IP network routers need to be modified.

5 Implementation

For fast prototyping and simple deployment, we have implemented AVES for reusable-IP networks as a suite of user-level software on the Linux platform. The three components are (1) the AVES-aware DNS server daemon, (2) the AVES waypoint daemon, and (3) the AVES NAT gateway daemon. To enhance security, data and control messages between the three components are authenticated by including with a message the 16-byte MD5 checksum [22] of the message together with a 16-byte secret key. One secret key is shared between the AVES-aware DNS servers and waypoints while each AVES-aware NAT gateway has a specific secret key. We save the discussion of some safeguarding security features until Section 6.2. In the following, we describe the three individual components, then we report some performance figures. Finally, we describe our current prototype system.

5.1 AVES-Aware DNS Server Daemon

Our AVES-aware DNS server daemon is based on the `named` DNS server in the BIND 8.2.3 distribution [11] and runs on a Linux PC. We modified `named` to intercept any outgoing DNS reply message containing a DNS name with the `avesnet.net` suffix because such a reply contains the reusable-IP address of the named responder. This is accomplished by inserting a function call in `ns_req()` (to intercept answers from the local cache) and `ns_resp()` (to intercept answers from other origins). Once a reply is intercepted, a lookup table is consulted to obtain the IP address of the reusable-IP domain's NAT gateway and a waypoint IP address is chosen. NAT gateway IP addresses are obtained from the NAT gateways periodically to accommodate dynamic address assignment (see Section 5.3 for more details), while the waypoint IP addresses and the reusable-IP host addresses are kept in configuration files. A SETUP message with a unique serial number is then sent via UDP to the chosen waypoint, the intercepted DNS reply is altered to contain the chosen waypoint IP address and is set aside. When the corresponding ACCEPT message is received from the waypoint, the DNS reply is finally sent to the requester. DNS replies that have been set aside are removed if the corresponding ACCEPT messages are not received within 3 seconds.

5.2 AVES Waypoint Daemon

Our AVES waypoints are based on Linux PCs. Each machine can be assigned multiple waypoint IP addresses as aliases of its network interface. The AVES waypoint daemon uses the Linux IP firewall (`ipfw`) API to filter selected data packets to user-level for manipulation, it requires Linux kernel version 2.2 or higher. To filter incoming data packets to user-level, the waypoint daemon opens a raw NETLINK_FIREWALL netlink socket. Filter entries can then be added to the input firewall via the `ipfw` API and the kernel can be instructed to direct matching packets to the netlink socket. After data packets are manipulated in user-level, they are reinjected into the network via a raw socket with the IP header included option (IP_HDRINCL) enabled.

We have fully implemented the delayed binding technique as described in Section 3.3.2. When there are multiple alias waypoint IP addresses on the machine, each address is treated independently by the waypoint daemon. The wait period T_{wait} in our implementation is 2 seconds which should provide sufficient time for a connection to be made. When the waypoint IP address IP_W is in a wait state, the waypoint daemon filters all in-coming packets with destination address IP_W regardless of the source address. Packets that do not indicate a new connection are processed normally according to existing translation table entries. A new connection (indicated by a TCP SYN packet, or any non-TCP packet) to IP_W is either accepted or rejected according to the algorithm shown in Figure 6. If the connection is accepted, a filter for the source and destination address pair is added to the firewall and a translation table entry is created. The packet is then processed normally. If the connection is rejected, the packet is dropped, and an ICMP "destination host unreachable" message [19] is sent back to the initiator. This signals to the initiator that

it needs to retry the connection. The reject period T_{reject} is 3 minutes in our implementation, which we think is sufficient to prompt the initiator to retry the connection, and does not make IP_W unavailable to the initiator again for too long. Note that, when IP_W is in a wait state, AVES SETUP messages sent to IP_W are ignored for simplicity. Below is a summary of the other noteworthy features supported by the waypoint daemon:

Fragmentation & Path MTU Discovery – Because the waypoint daemon encapsulates a translated packet in an IP header and adds a 16-byte MD5 checksum, typical 1500 byte in-coming Ethernet packets will have to be fragmented on their way out. It turns out that Linux does not perform fragmentation for packets sent through a raw socket with the IP_HDRINCL option enabled, therefore IP fragmentation has been implemented in the waypoint daemon. The waypoint daemon also supports path MTU discovery [16]. That is, when the "Don't Fragment" flag of an in-coming IP packet is set but fragmentation is necessary, the waypoint daemon drops the packet, and returns an ICMP "destination unreachable fragmentation needed" message [19] to the initiator with the MTU field set to 1464 bytes. Finally, a consequence of IP fragmentation is that, the AVES NAT gateway must be configured to reassemble all in-coming fragmented packets so that the AVES NAT daemon can function properly.

Protocol Specific Timeouts – A translation table entry represents a session opened by an initiator and will expire if there is no traffic activity for a period of time. To optimize resource usage, we use different timeout values for different protocols. The protocols the waypoint daemon recognizes are ICMP, TCP, and UDP. First, if an initiator is transmitting an unknown protocol or a mixed set of protocols to the responder, a default timeout value of 15 minutes is used. For ICMP, since it is mostly generated by `ping` or `traceroute`, we aggressively timeout these entries in 1 minute. For UDP, the timeout value is set to 15 minutes. For TCP, the timeout value is set to 30 minutes. These choices are somewhat arbitrary, but we think they are reasonable. To further optimize, we keep track of the TCP connections that correspond to a translation table entry, and when all of them have terminated (indicated by TCP FIN packets), the translation table entry is removed immediately without waiting for the timeout. An exception to this is when the traffic is HTTP (i.e. port 80) because popular browser software such as Netscape and Internet Explorer always cache DNS replies for 15 minutes. Thus, for HTTP, we simply use a timeout value of 20 minutes without checking for TCP FIN packets.

5.3 AVES NAT Daemon

Our AVES-aware NAT gateways are based on Linux PCs as well, and they are assumed to be already configured to perform defragmentation of in-bound packets and IP masquerading (i.e. out-bound NAT), which is fully compatible with AVES. Similar to the waypoint daemon, the AVES NAT daemon also filters selected packets to user-level for manipulation. To handle NAT gateway dynamic IP address assignment, periodically, the NAT daemon sends authenticated registration messages via UDP to the AVES-aware DNS severs to report its current IP address. These messages are sent more frequently when an address change is detected to ensure with high probability that the update is completed promptly.

The basic operations performed by the NAT daemon is as described in Section 3.2. The NAT daemon by default filters all in-coming encapsulated packets. When an authentic encapsulated packet is received and the connection has not been seen before, a filter is installed for the corresponding out-bound packets, and a translation table entry is created. Several other noteworthy features of the NAT daemon are summarized below:

Protocol Specific Timeouts – Similar to the waypoint daemon, different timeout values for the translation table entries are used for different protocols. The policy is exactly the same as that in the waypoint daemon.

ICMP Handling – For `traceroute`, even though the in-bound packet is UDP, an out-bound ICMP packet is triggered. To support `traceroute`, when an in-bound UDP connection is received, we install an extra filter and translation table entry for the potential out-bound ICMP packets. The timeout is set to 5 seconds so that if no ICMP packets are triggered, the state is removed quickly. In addition, since many ICMP message types carry IP addresses and port numbers in the packet payload, the AVES NAT daemon translates the payload accordingly as well.

Multi-Homing Support – To support multi-homing as described in Section 3.2, the source address and port number (or ICMP Identifier) of an in-bound packet are translated. To choose a suitable port number, we simply pick a port number between 1024 and 65535 at random, and test to see if that port number can be bound to a TCP and a UDP socket. The process repeats until a port number that is free is found. This makes sure that our port number allocation will not interfere with the other operations of the NAT gateway. However, notice that our straight-forward implementation does not achieve the theoretical highest connectivity as discussed in Section 4. In our implementation, only 64,512 in-bound connections (TCP *or* UDP) can be simultaneously active regardless of the destinations of the connections. This is however more than sufficient for the purpose of our prototype.

Limitations of Multi-Homing Support – When multi-homing is enabled, only one reusable-IP network can be connected to a NAT gateway because when there are multiple reusable-IP networks attached, our implementation is not yet capable of translating the source address of an in-bound packet to the address of the correct output network interface. Also, some applications, most notably `ftp`, will not work when multi-homing is enabled because the

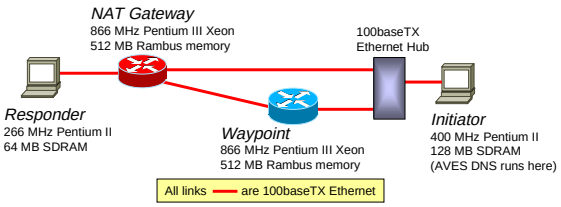

Figure 7: Performance measurement testbed

reusable-IP `ftp` server cannot open the data connection to the IP client since the IP client's address has been translated. And because the server passes its reusable-IP address, say 10.0.0.1, to the client, even if "passive mode" [1] is enabled, the client will attempt to open the data connection to the address 10.0.0.1 instead of the waypoint IP address. The only way to get around this is to translate `ftp` control packets' content.

5.4 Performance

To measure the performance of our system, we set up a small 100 Mbps Ethernet testbed as shown in Figure 7. For data path performance, we instrumented the Linux kernel version 2.2.14 and our daemon software to measure, with the Pentium CPU cycle counter, the processing time of a packet in the waypoint and the NAT gateway (both inbound and out-bound directions). We measured three quantities: (1) the total packet processing time from the moment `netif_rx()` was called by the Ethernet device driver after receiving a packet until the moment `dev_queue_xmit()` was called to pass a processed packet to the device driver for transmission; (2) the AVES daemon processing time from the moment a packet was received by a daemon socket until the moment before the processed packet was sent out on a socket; (3) the time spent on computing the MD5 authentication checksum in the AVES daemon. We sent UDP packets of varying sizes between the initiator and the responder and recorded the processing times. Our experiments show that all the processing times scale linearly as the packet size varies. Figure 8 shows the partial results, averaged over 10,000 packets, for the smallest (36 bytes) and largest (1464 bytes) packet sizes we have tried.

There are several noteworthy points. First, implementing our software in user-level adds a very significant overhead due to the memory copies and context switches. We can expect a kernel-level implementation of our software will have a total processing time very close to the AVES daemon processing time. Second, almost all the AVES daemon processing time is spent on computing the MD5 authentication checksum (note that no authentication is needed for out-bound packets at the NAT gateway). This overhead can be reduced if we only authenticate the packet headers but at the cost of lowered security. Finally, based on these measurements, our software can theoretically sustain a throughput of 233 Mbps with 1464 byte packets in our testbed.

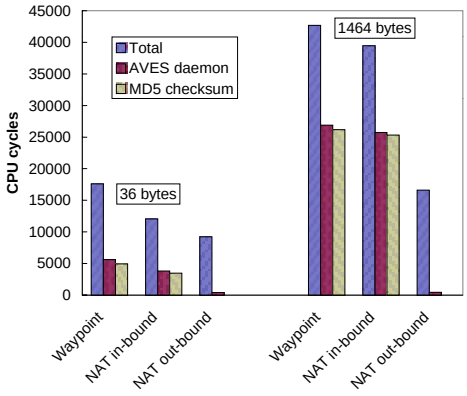

Figure 8: Packet processing times

We have also conducted end-to-end throughput experiments. When we sent 1464 byte packets from the initiator to the responder, the throughput was limited only by the link capacity, as the system achieved 96 Mbps with UDP and 80 Mbps with TCP. However, when we sent 48 byte packets, our software was only able to achieve 41 Mbps with TCP. This is actually higher than the calculated maximum of 19 Mbps based on the processing times measurements due to the amortization of kernel overheads over a sequence of packets. We expect the throughput with UDP to be slightly better; however, due to a device driver bug with the Intel EtherExpress Pro 100 network interface card, we were unable to send 48 byte UDP packets faster than 10 Mbps without causing the interface card to shutdown.

Next we measured the performance of the control path. Typically, the time required to resolve an AVES DNS name is dominated by the network delays of the DNS and AVES control messages. To factor out the network delays, we ran both the AVES-aware DNS server daemon and the waypoint daemon on the initiator machine. A program running on the initiator that repeatedly issued `gethostbyname` system calls for the responder was used to drive the system. We then measured the number of CPU cycles, including socket reads and writes, averaged over 20 requests, consumed by each control path component.

We found that the total time required to complete a `gethostbyname` system call was on average 357,000 cycles. This total time can be further broken down as follows. First, it took 142,000 cycles to process the DNS query at the AVES-aware DNS server daemon and send the SETUP message to the waypoint daemon. Second, the waypoint daemon took 71,000 cycles to process the SETUP message and send back the ACCEPT message. Finally, another 17,900 cycles were spent at the AVES-aware DNS name server daemon to process the ACCEPT message and send out the final DNS reply message. Computing the MD5 authentication checksum of an AVES control message took 3,100 cycles. Thus, the DNS query processing is the bottleneck. With a 400 MHz AVES-aware DNS server, at most 2,800 sessions can be set up per second. Of course

if delayed binding is used, the protocol will impose a much stricter limit.

5.5 Prototype System

We have registered the domain name suffix `avesnet.net` and deployed an AVES prototype system. A Linux PC serves as the AVES-aware DNS server for the `avesnet.net` domain. Two other Linux PCs serve as waypoints, each with 25 IP aliases for a total of 50 waypoint IP addresses. We currently have ten trial subscribers. Subscribing to AVES is a simple three step process. A reusable-IP network operator needs to (1) obtain a subdomain under `avesnet.net` from the service operator, (2) inform the service operator the desired DNS name to reusable-IP address mappings, and (3) run the AVES NAT gateway daemon.

Using our prototype, we have shown that a diverse set of applications work seamlessly with AVES. We are able to remote login from any IP host to a demo reusable-IP host called `demo1` using `telnet` or `ssh`, perform file transfers using `ftp` (when multi-homing is disabled, using non-passive mode) or `scp`, export a NFS file system on `demo1` and mount the file system on any IP host. We are also able to host a web server on `demo1` and access the content from any IP host. When logging in from `demo1` to an IP server (by out-bound NAT), we are able to directly bring up `X Windows` applications on `demo1` after the DISPLAY environment variable has been correctly set. Diagnostic tools such as `ping` and `traceroute` also work transparently (with limitations described in Section 6). An on-line demo of our prototype can be found at `http://www.cs.cmu.edu/~eugeneng/research/aves/`.

6 Discussion

In this section, we discuss the limitations imposed by our approach. This is by no means an exhaustive account. It is important to realize that AVES is making a trade-off between non-intrusively restoring bi-directional connectivity to a high degree and the limitations it imposes. We believe that most of the limitations can be coped with, and the benefits of AVES significantly out-weigh the limitations.

6.1 Application Requirements

There are three types of limitations imposed by AVES that may conflict with an application's behavior, they are (1) limitations due to address translation, (2) limitations due to the need for session creation, and (3) limitations due to the need for consistent state maintenance. In the following, we discuss what rules must an application obey in order to be compatible with AVES.

The first type of limitation is not specific to AVES, but it is a fundamental limitation of any address translation scheme such as NAT, TRIAD [3] and IPNL [6] (discussed in Section 7). The main problem is that some applications *break* the layering semantics by exchanging lower layer information such as IP addresses and use the information directly. In [10] and [24], some NAT-friendly application design guidelines are given. Because AVES also performs address translation, some of these guidelines are relevant (guidelines that aim to avoid in-bound connections are no longer needed under AVES). Specifically, with respect to address translation, in order to be compatible with AVES coupled with NAT, an application should not pass IP addresses in the packet payload; instead, DNS names should be passed, and name resolution should always be used to determine the IP addresses. Listener port number passing is actually no longer a problem if DNS names are used. Also, applications should not expect the network and transport headers to be unmodified in transit. Clearly IPsec [13] would not work across NAT or AVES. In IETF, there is ongoing work on making NAT more IPsec-friendly [2]. In Section 6.4, we will also describe a change to the AVES data path that may make AVES more IPsec-friendly.

AVES fundamentally requires a session to be opened by an initiator before connectivity is provided. Therefore, an application must perform a DNS lookup before communication begins. Moreover, communication must begin immediately after the DNS lookup to work with delayed binding since T_{wait} is typically small. When a connection is rejected, the application must perform a DNS lookup again to restart the session. Note that a T_{wait} of 2 seconds used by our prototype might not work for an application like `traceroute`, since it progressively probe the network hop-by-hop and this process may take more than 2 seconds to reach the waypoint.

Finally, an application must obey some rules to maintain consistency between its state and the waypoints' state. From a waypoint's point of view, a session is terminated when an idle timeout occurs, or when all connections of the session (assuming they are all TCP) are terminated. Therefore, an application must send periodic keep-alive messages. In addition, it must not reuse DNS lookup results across sessions (as in the web browser example). An application must also begin communication within T_{wait} after a DNS reply is received, otherwise, the application's view is stale. When a connection is rejected, an application must also restart the session by performing another DNS lookup. These rules will prevent the initiator from having a stale view. If an application does not follow these rules, then it may have a stale view, in that case, there are two possible outcomes. First, the connection may get rejected by the waypoint because it has no state for the initiator. Second, the connection may get relayed to the wrong responder because the waypoint has other state for the same initiator. On the other hand, a waypoint may have a stale view if

a session has ended (e.g. a UDP session is terminated by the application) but it still keeps state about it. This type of inconsistency only affects performance, not correctness, because it simply makes the waypoint unavailable to the same initiator for a longer period.

6.2 Security

An obvious concern with AVES is whether it is secure. Can attackers flood the system? Will AVES reusable-IP hosts be exposed to attackers at the level of regular IP hosts? Can attackers cause the system to mis-behave? In the following, we discuss these issues in detail. We assume a general deployment scenario where delayed binding is used since a secure environment is assumed in intranet deployment. To summarize, the connectivity to AVES reusable-IP hosts is more easily disrupted by flooding attacks than that to regular IP hosts, however, AVES reusable-IP hosts are somewhat less vulnerable to other security exploits. Attackers also cannot cause waypoints to incorrectly relay traffic.

First and foremost, we acknowledge that AVES waypoints are no better at handling packet flooding type of denial of service attacks than any other network systems. The only method to prevent this is to traceback to the origin of the flooding and filter those packets out of the network. There has been some recent advances in this area [23]. Ingress filtering [5] also helps reduce the problem by disallowing address spoofed packets from entering the network. When the waypoints are flooded, reusable-IP networks will only have out-bound connectivity through NAT as in without AVES. In our implementation, we simply put some hard limits on resource consumptions to prevent overloading of each AVES component during a flooding attack. At a different level, to cope with aggressive users, the AVES-aware DNS server can potentially be extended to allocate the available session creation capacity more fairly by scheduling requests based on the initiators' and responders' identities. This way, an initiator or a responder (e.g. a popular web server) cannot occupy all resources and prevent other normal users from opening sessions. Currently, our implementation simply limits the peak rate at which sessions can be opened to each responder.

To address the second question, although AVES provides in-bound connectivity, it does not fully expose reusable-IP hosts and attacking them is somewhat more difficult. We have disabled the zone transfer [15] function of the AVES-aware DNS server to prevent malicious users from obtaining host names. In addition, to prevent scanning of host names, our implementation ignores and penalizes a requester that queries for host names that do not exist in our database. Without knowing any host name, the only opportunity for an attacker to connect to a reusable-IP host is to transmit packets to a waypoint during the time it is in a wait state. To lower the chance of this succeeding, our waypoint daemon monitors for in-coming packets with source addresses that it has no state for while it is not in a wait state and reject all packets from these sources for 3 hours.

Finally, an attacker may hope to cause waypoints to misbehave by sending malicious packets to a waypoint while it is in a wait state. However, we have designed the wait state algorithm such that these malicious packets *cannot* cause a waypoint to mis-behave, they *cannot* prevent a legitimate initiator from connecting to the correct responder. The reason is that the wait state period is fixed and does not end simply because a malicious new initiator has arrived. The rejection algorithm is also conservatively designed to make sure all admission control violations are caught even in the presence of malicious packets.

6.3 Scalability

Because AVES is optimized for deployment, its scalability is a key concern. First, on the control path, since the AVES-aware DNS server can be replicated easily, DNS query processing should not present scalability problems. For intranet deployment, when local DNS server upgrades are possible, there is no protocol imposed limit on the rate at which sessions can be opened, and we have shown that a Linux PC waypoint can process thousands of requests per second. However, if the delayed binding technique is used, the rate at which the system can accept sessions is limited by the protocol. For our prototype system, 25 sessions can be accepted per second. Under such constraints, AVES should not be used to serve a busy web server. Note that this session acceptance rate limit *does not* reduce the connectivity achievable by the system as stated in Section 4.

On the data path, the scalability concern is whether the service provider's waypoints can handle the data traffic from initiators. Our experiments have demonstrated that our un-tuned implementation of AVES achieves a reasonable level of performance. With the advances in tera-bit class router technologies, we believe the data path operations can be performed at very high-speed. An alternative approach is to harness the resources of the NAT gateways of AVES subscribers, and use these NAT gateways as waypoints to relay subscribers' traffic. This way, the number of waypoints increases with the number of AVES subscribers, ensuring scalability. Although our software can be extended easily to support this service model, it introduces several new problems. Since waypoints are no longer owned by a trusted service provider, it is not clear what type of security protection can be achieved. Also, because waypoints can no longer be assumed to be always-on, maintaining the set of waypoints dynamically and providing fault tolerance are important problems to be addressed.

6.4 Potential Extensions

IPv6 Support – Our implementation currently does not support IPv6 header conversion, this is an important extension that is needed.

Coexisting with Ingress Filtering – Consider the example in Figure 4 again. In step 5, R is effectively spoofing IP_W. This is done for simplicity and performance reasons. Routers that implement ingress filtering [5] will drop such packets. AVES can easily be enhanced to work with ingress filtering by making R tunnel the packet to W, and let W forward the packet to A. The disadvantage is that the load on W is increased.

Coexisting with IPsec – To make NAT IPsec-compatible, RSIP [2] has recently been proposed in the IETF. In order for AVES to be compatible with IPsec, packet content must not be altered in transit. This can be achieved if the responder is made aware of the fact that it is being virtualized by a waypoint. This idea is in-spirit similar to that in RSIP. Using the example in Figure 4 again, the waypoint can generate the packet $[IP_W \rightarrow IP_R[IP_A \rightarrow IP_W]]$ (step 2), R can forward the packet $[IP'_R \rightarrow IP'_B[IP_A \rightarrow IP_W]]$ (step 3), and the responder itself can generate the packet $[IP'_B \rightarrow IP'_R[IP_W \rightarrow IP_A]]$ (step 4). The reusable-IP responder now needs to be heavily modified, although there are some incentives to do so.

Connectivity for Non-IP Initiators – AVES is designed to solve the connectivity problem of cases (a) and (b) in Table 1. Since other cases are reducible to either case (a) or (b), AVES functions correctly in all cases. However, because AVES perceives all non-IP initiators belonging to the same non-IP network as a single IP initiator (since they are masked by their NAT or NAT-PT gateway), the connectivity provided by AVES to each individual non-IP initiator is correspondingly reduced. Precisely, with N IP addresses allocated for AVES waypoints, each non-IP *network* can simultaneously reach up to N non-IP responders. Although the connectivity is reduced, it is important to realize that this is perhaps the best one can achieve if the initiating non-IP network has no incentive to make any upgrade. If upgrading is acceptable, higher connectivity for these cases can be achieved by extending the NAT or NAT-PT gateways to implement a more sophisticated solution such as TRIAD [3] or IPNL [6]. A discussion on TRIAD and IPNL can be found in Section 7.

7 Related Work

In this section, we first review some well known partial work-arounds to cope with the lack of in-bound connectivity. Then we discuss a solution that is currently proposed in the IETF. Finally, we discuss other related work that are not directly addressing the in-bound connectivity problem.

A common work-around for the lack of in-bound connectivity is to forward a port number of the NAT gateway to a specific host inside the reusable-IP network. For example, in-coming traffic to port 23 (i.e., `telnet`) of the NAT gateway can be blindly redirected to port 23 of a particular reusable-IP host. With this transport layer work-around, although more than one reusable-IP host is reachable, no two reusable-IP hosts can offer the same service (e.g. no two reusable-IP host can simultaneously support port 22 `ssh` login), thus unacceptable connectivity is provided. Tedious per application manual configurations are also required.

A related technique is to take advantage of a new type of DNS resource record proposed in [9], called the SRV resource record, which can specify the port number of a service. When both the service provider *and* the client support DNS SRV, the client can retrieve both the IP address of the host that is offering the service and the exact port number it should use to use the service. Suppose the port number binding can be dynamically assigned by a NAT gateway, then better in-bound connectivity to non-IP hosts can be achieved than port number forwarding. Unfortunately most applications and operating systems today do not support this feature.

Another work-around exists for UDP communication. Assume both the initiator and the responder are behind NAT gateways. The idea is to have both the initiator and the responder contact an IP server to exchange their NAT gateways' IP addresses, then both initiator and responder simultaneously send each other UDP packets with the same source and destination port numbers. Assuming the NAT gateways do not alter the source port numbers of these packets, bi-directional communication can be achieved. This work-around has been applied to some networked games [12]. Note that this scheme only works for UDP, requires a third party connection broker, and both parties must be actively involved, which is not suitable for client-server applications.

Another possibility is to insert a globally unique host name into packets so that a NAT gateway can dynamically determine the destination of a packet by looking up the host name. Host Identity Payload [17], proposed in the IETF, may be used for this purpose. Existing IP hosts or edge routers must however be modified to insert such host names into packets. With HTTP/1.1 [7], it is possible to embed the name of the destination in the HTTP header. This technique has been used to perform HTTP virtual hosting. This is however not a general solution for applications that are not based on HTTP.

Recently, a solution based on the SOCKS protocol has been proposed in the IETF [14]. The idea is that, when an application performs a DNS lookup for a responder, a "fake" IP address (e.g. 0.0.0.1) is returned to the application. When the application actually makes a socket call to communicate with the "fake" IP address, the SOCKS library on the initiator intercepts the call and connects to the SOCKS server on the responder's NAT gateway. The DNS name of the responder is communicated to the SOCKS server, and the SOCKS server connects to the real responder. Data packets are then copied between the two spliced connections at the NAT gateway. The downside of this scheme is that existing IP hosts need to be upgraded. It is conceivable that the initiator-side's SOCKS processing can be pushed to the initiator's edge router; in that case,

existing edge routers need to be upgraded.

Next we discuss two on-going research projects that are closely related to AVES. In [3], Cheriton *et al.* propose a solution called TRIAD that can solve the IP address scarcity problem. TRIAD makes it possible to expand the Internet by arbitrarily connecting an unlimited number of IP network realms, each with its own 32-bit address space. TRIAD uses DNS names rather than addresses for global identification. During DNS name resolution, a sort of realm-to-realm source route is computed. A simple "shim" protocol header is added to every packet to carry this realm-to-realm source route to assist routing across multiple realms.

IPNL [6] is another recent proposal to provide an alternative to IPv6. IPNL also uses DNS names as global identifiers and allows multiple IP realms to be connected. However, rather than allowing IP realms to be connected arbitrarily as in TRIAD, IPNL allows IP realms to be organized hierarchically, with a single global "middle realm" and many smaller realms connected to the "middle realm". This allows IPNL to have better routing efficiency compared to TRIAD. IPNL introduces two extra levels of optional headers to permit communication across realms. To communicate, the initial packet contains the DNS names of the source and the destination. As the packet traverses the realms, various addresses are resolved and stored in the packet headers. These addresses are then used for fast packet forwarding and the DNS names can be omitted.

While the goal of TRIAD and IPNL is to provide an alternative to IPv6, the goal of AVES is to maintain connectivity between today's IP Internet and emerging networks of IPv6 and reusable-IP address spaces. In contrast to AVES, TRIAD and IPNL only allows hosts within realms running those respective protocols to communicate with each other. However, unlike TRIAD, AVES cannot route packets over an arbitrary number of IP networks, nor can AVES achieve the level of connectivity of TRIAD or IPNL. Nevertheless, we believe that maintaining connectivity between existing IP hosts and IPv6 or reusable-IP hosts is an important problem, therefore the trade-off is justified.

8 Summary

The main contribution we make in this paper is that we propose a waypoint service called AVES that can provide high connectivity from IP hosts to IPv6 or reusable-IP hosts without consuming many IP addresses or changing existing IP hosts and IP network routers. AVES is optimized for deployability and can be deployed easily as a 3rd-party network service. We have implemented and deployed a prototype system at CMU, and have received very positive feedbacks from our subscribers. Further information on AVES can be found at `http://www.cs.cmu.edu/~eugeneng/research/aves/`.

References

[1] S. Bellovin. Firewall friendly FTP, February 1994. RFC-1579.

[2] M. Borella, J. Lo, D. Grabelsky, and G. Montenegro. Realm Specific IP: A framework, July 2000. Internet Draft, draft-ietf-nat-rsip-framework-05.txt.

[3] D. R. Cheriton and M. Gritter. TRIAD: A new next generation Internet architecture, March 2000. http://www-dsg.stanford.edu/triad/triad.ps.gz.

[4] S. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) specification, December 1998. RFC-2460.

[5] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, May 2000. RFC-2827.

[6] P. Francis. IPNL architecture and protocol description, May 2000. Available from the author.

[7] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC-2616.

[8] R. Gilligan and E. Nordmark. Transition mechanisms for IPv6 hosts and routers, April 1996. RFC-2893.

[9] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV), February 2000. RFC-2782.

[10] M. Holdrege and P. Srisuresh. Protocol complications with the IP network address translator, January 2001. RFC-3027.

[11] Internet Software Consortium. Berkeley Internet Name Domain (BIND) version 8.2.3. http://www.isc.org/products/BIND/.

[12] D. Kegel. NAT and peer-to-peer networking. http://www.alumni.caltech.edu/~dank/peer-nat.html.

[13] S. Kent and R. Atkinson. Security architecture for the Internet Protocol, November 1998. RFC-2401.

[14] H. Kitamura. A SOCKS-based IPv6/IPv4 gateway mechanism, March 2001. Internet Draft, draft-ietf-ngtrans-socks-gateway-06.txt.

[15] P. Mockapetris. Domain names - concepts and facilities, November 1987. RFC-1034.

[16] J. Mogul and S. Deering. Path MTU discovery, November 1990. RFC-1191.

[17] R. Moskowitz. Host Identity Payload, February 2001. Internet Draft, draft-moskowitz-hip-arch-02.txt.

[18] E. Nordmark. Stateless IP/ICMP translation algorithm (SIIT), February 2000. RFC-2765.

[19] J. Postel. Internet Control Message Protocol, September 1981. RFC-792.

[20] J. Postel. Internet Protocol, September 1981. RFC-791.

[21] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets, February 1996. RFC-1918.

[22] R. Rivest. The MD5 message-digest algorithm, April 1992. RFC-1321.

[23] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of ACM SIGCOMM'00*, Stockholm, Sweden, Aug. 2000.

[24] D. Senie. NAT friendly application design guidelines, March 2001. Internet Draft, draft-ietf-nat-app-guide-05.txt.

[25] P. Srisuresh and K. Egevang. Traditional IP network address translator (Traditional NAT), January 2001. RFC-3022.

[26] P. Srisuresh, G. Tsirtsis, P. Akkiraju, and A. Heffernan. DNS extensions to network address translators (DNS_ALG), September 1999. RFC-2694.

[27] G. Tsirtsis and P. Srisuresh. Network address translation - protocol translation (NAT-PT), February 2000. RFC-2766.