IPNL: A NAT-Extended Internet Architecture*

Paul Francis
Tahoe Networks

Ramakrishna Gummadi
UC Berkeley

ABSTRACT

This paper presents and analyzes IPNL (for IP Next Layer), a NAT-extended Internet protocol architecture designed to scalably solve the address depletion problem of IPv4. A NAT-extended architecture is one where only hosts and NAT boxes are modified. IPv4 routers and support protocols remain untouched. IPNL attempts to maintain all of the original characteristics of IPv4, most notably address prefix location independence. IPNL provides true site isolation (no renumbering), and allows sites to be multi-homed without polluting the default-free routing zone with per-site prefixes. We discuss IPNL's architectural benefits and drawbacks, and show that it comes acceptably close to achieving its goals.

1. INTRODUCTION

The IP architecture has undergone steady change over the last 10 years or so. The most significant change has been the introduction and spread of NAT (Network Address Translator) [9], and the resulting loss of end-to-end addressability. Despite this loss, the Internet has seen, and, we believe, will continue to see nearly all of its tremendous growth in a NAT'ed world. How can this be, if end-to-end addressability is so fundamentally important to the Internet?

The answer is "obvious"—the dominant applications of the Internet, such as email and web, are client/server applications. Only the servers need to be globally addressable. As long as the servers are not behind NATs, they are addressable and everything can be made to work. The kinds of applications that are disabled by NAT, including so-called peer-to-peer applications, simply have not been important economic drivers of the Internet.

But this cannot be all of the answer. It isn't enough to say that NAT has spread because it doesn't break client/server applications. NAT must also be providing some benefits.

1.1 Pros and Cons of NAT

The main obvious benefit of NAT is that it expands the IPv4 address space. The second important benefit of NAT is that it isolates a site's address space from the global address space[1]. This address isolation benefits both the provider and the subscriber. A site behind a NAT box can change providers without having to renumber internal hosts. Likewise, providers can freely renumber their networks for better address aggregation. We suspect that address isolation is one of the primary reasons that NAT is as popular as it is, and that the IETF has underestimated the appeal and importance of this feature.

Because of this address isolation, a NAT'ed site can be attached to multiple ISPs (Internet Service Providers) without having the site's address prefix advertised across the default-free routing zone of the Internet. Because of this, it can be argued that NAT is a key technology responsible for what limited scalability the Internet has.

The primary negative aspect of NAT is that it inhibits the introduction of certain kinds of peer-to-peer applications. It does this in two ways. First, a host behind a NAT box is not generally addressable from the global Internet, at least not in the traditional pre-NAT way. Second, some peer-to-peer applications fail to work properly in the face of address translation or port translation. These applications require application layer gateways in the NAT boxes. An example of such an application is SIP (Session Initiation Protocol) [13].

A second negative aspect of NAT is that it complicates scalable network operation and new protocol and application design. Everything must now take into consideration the various possible NAT deployments, of which there are many. So while it is possible to address hosts behind NAT boxes, doing so is much more complicated than it would be with global IP addresses. For instance, a given port at the NAT box can redirect packets to a given address and port. Or, the NAT box can snoop SIP, assign port translations, and modify SIP messages accordingly.

A significant part of the problem with NAT is that there has been little effort on the part of the Internet community to design and standardize ways of interoperating with NAT boxes, and making them easier to deal with. When the first commercial NAT products were being sold, the IAB (Internet Architecture Board) and IETF took a "just say no" attitude towards NAT, assuming that IPv6 [6] would take over before NAT spread too far. It is only this year, a decade after NAT was first published, that the IETF has decided to work to improve interoperating with NAT[2].

1.2 An Extension to NAT

Given the popularity of NAT, its success at mitigating the ad-

*Work done while both authors were at ACIRI.

[1] A site is defined loosely as a private IP network that connects to the Internet via one or more providers. A site may be as large as a global enterprise network or as small as a single home.

[2] The midcom (middlebox communication) working group is chartered with the task of enabling communications between applications and NATs and firewalls. This is in contrast with the several years old NAT working group, which was primarily chartered with documenting NAT and the problems associated with it.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

dress depletion and scaling problems of IPv4, the existence of an economic driver for its deployment, and the simple fact that the Internet has thrived on a NAT'ed architecture, we were motivated to ask whether some extension of NAT wouldn't make for a genuinely suitable Internet architecture. That is the question addressed by this paper.

This question leads to two more questions:

1. What constitutes a suitable Internet architecture, and
2. What constitutes an extension of NAT (versus a whole new Internet protocol)?

We have chosen to define a suitable Internet architecture as one that preserves the original characteristics of IPv4 while solving its scalability and address depletion problems. These characteristics include:

1. All hosts have long-lived, globally routable addresses (if they so choose) that serve to also identify the host.
2. Routers are stateless—they do not require per-connection state, and failure in a router does not result in failure of the connections[3] going through them if an alternate path exists.
3. A network's address prefix is assigned independently of where the network attaches to the Internet.
4. Packets cannot be easily hijacked by rogue or misconfigured hosts that are not on the physical path of the packets. In IPv4, this characteristic derives from the fact that addresses are both locators and identifiers, and that routers enforce packet delivery to the right location through hop-by-hop coordination and trust.

We emphasize that, in maintaining the characteristics of the original IP while improving its scalability and size, we are, in essence, trying to design a better protocol than IPv4.

Regarding the second question, we define an extension of NAT as one that works by modifying only hosts and NAT boxes. In particular, existing IPv4 routers must not change, and, by extension, the IPv4 layers of hosts must also not change. We also require no changes to DNS, and no new global addressing authorities.

By allowing changes to hosts, we break one of the fundamental properties that has allowed NAT to succeed—that it can be deployed without changes to hosts. This seems to us to be inevitable. We cannot think of any way to significantly improve NAT without making changes to the host. Having said that, we can make the observation that not all host changes are equal. In particular, changes above the transport layer are easier to deploy than changes below the transport layer. Because we have chosen to maintain the original characteristics of IPv4, we have been compelled to make our changes below transport and above IPv4.

An alternative approach would be to forego trying to maintain the original characteristics of IPv4, but allow simpler changes to hosts. For instance, SIP provides end-to-end addressability in the face of NATs, as long as the NAT boxes are SIP proxies or tightly coordinated with SIP proxies. This NAT-as-SIP-proxy approach is stateful, unscalable, has a costly setup, and is a single point of failure (failure recovery would require additional changes to applications). Therefore, it loses some of the important characteristics of the original IPv4 architecture.

Unless IPv6 gets deployed, however, it seems likely that a SIP-like approach, where the host communicates with the NAT box and the firewall out-of-band[4], is the next likely step in the evolution of the Internet.

[3]The term connection in this paper refers to data flow between transport layer entities.

1.3 IPNL: IP Next Layer

This paper presents a NAT-extended protocol called IPNL, for IP Next Layer. The purpose of pursuing a totally new design at this relatively late stage in the IPng process is not so much to try to supplant IPv6. Rather, we wish to shed light on a number of questions that remain relevant a full decade after the first wave of IPNG proposals, including a number of NAT-extended proposals, documented in RFC1380 [12].

These questions include:

1. Can a NAT-extended protocol achieve the original characteristics of IPv4, and therefore serve as a long-term architecture?
2. Can the router scaling problem be solved while maintaining the original IPv4 characteristic of addressing independence?
3. Is a NAT-extended approach less expensive than a full replacement of IPv4?

We believe the answer to the first two questions to be a qualified "yes". The original characteristics of IPv4 include long-lived addresses, robustness/statelessness, address independence, and packet hijacking resistance. As will be shown in the remainder of this paper, IPNL is somewhat inelegant in its approach to robustness and hijack resistance. In both cases, additional complexity in the host's IPNL layer is required to bring IPNL close to IPv4 in those areas. Except for that, we believe that we have succeeded in our goals.

As for the third question, more experience with IPNL would be required to determine if it is less expensive than IPv6. In particular, the full set of protocols (routing protocols, mobility protocols, multicast protocols, etc.) would have to be specified in order to compare the two side-by-side. Nevertheless, we make some arguments later on that suggest that a NAT-extended approach may be less expensive to deploy than IPv6.

The major attributes of IPNL are as follows:

- It is a NAT-extended architecture, which means that it maximizes reuse of the existing IPv4 infrastructure, primarily by adding a new layer above IPv4 that is routed by NAT boxes.
- It utilizes Fully Qualified Domain Names (FQDNs) as an end-to-end host identifier in packets.
- It extends the IP address space such that the globally unique IP address space forms the high order part of the IPNL address, and the private IP address space forms its low order part.
- It completely isolates site addressing from global addressing.

The rationale for these attributes is as follows:

Infrastructure reuse: We make the assumption that reusing an existing infrastructure lowers the cost of deploying a new protocol. This reduction in cost comes not just from continued use of the current IPv4 infrastructure (including the human skills needed to run it), but also from shrinking the number of phases for deployment of the new protocol from three to two. Specifically, deployment of IPv6 requires three (concurrent) phases: 1) IPv4 hosts talking to IPv6 hosts, 2) IPv6 hosts talking to each other tunneled over IPv4,

[4]Broadly stated, this is the approach being taken by the IETF midcom working group.

and 3) pure IPv6. Mechanisms to fully support all three phases and their interactions must exist.

By architecting IPNL as a layer above IPv4, we avoid the third phase. We assume that this significantly reduces complexity (two phases instead of three, one combination of phases instead of four). For example, approximately half of RFC3056 (6to4) [2] is devoted to interoperability between 6to4 and "native" IPv6. In other words, half of the complexity of RFC3056 comes from interactions between the second and third phases of IPv6 deployment. Having said this, we are quick to point out that we have not analyzed the costs of both approaches any deeper, so this is only an assumption at this point.

FQDN Utilization: The motivation behind using FQDNs also derives from an assumption of lowered deployment cost. In this case, the lowered cost comes from 1) not having to define and administer a new global address space, and 2) being able to reuse much of the existing support infrastructure and applications, including host configuration infrastructure (for example, DHCP [8]), AAA infrastructure (for example, RADIUS [20]), and SIP [13], all of which use FQDNs as the primary form of host identification. The use of the FQDN in this role, however, results in a somewhat different architecture, and the costs and potential weaknesses of this change must be considered.

Extended IP address space: This is a natural result of using the existing topology of private address realms connected to each other and the global IP Internet by NAT boxes. Again, by using existing addresses and topological components (realms and NAT boxes), we attempt to minimize deployment costs.

Isolated site addressing: This is the only major attribute that doesn't derive from an attempt to reduce costs. Rather, this attribute is the cornerstone of our approach to achieving global scalability in the face of multi-homed sites. The basic idea here is that if we can completely isolate site operation from issues of global connectivity, the ISPs are free to manage addresses as they see fit. The importance of this cannot be understated, and can be understood in light of the following thought experiment.

Consider the case where an ISP has, to pick a number, 50,000 home subscribers in a given address aggregation. The ISP would like to modify the prefixes of half of these subscribers in order to improve its aggregation. Now imagine the ISP having to coordinate with 25,000 home subscribers in order to carry out this change in prefix. Even with automatic renumbering mechanisms, it is highly likely that many things will go wrong in the home networks, making the whole process difficult and expensive. If, on the other hand, the home network addressing is isolated from global addressing, the change could be made without having to contact the home subscribers at all, and the whole process would be greatly simplified.

As part of site isolation, IPNL allows connections to survive renumbering and address change "events" during their lifetime (Section 3).

1.4 Outline

Because of space constraints, this paper does not give a complete description of IPNL. Rather, it focuses only on the key architectural aspects of IPNL. As a result, the reader will walk away from this paper with a basic understanding of IPNL, but also with a host of unanswered questions, some large, some small. For instance, this paper does not describe how IPNL does mobility, multicast, host auto-configuration, anycast, or interworking between legacy IPv4 and IPNL hosts, even though these are obviously critical requirements. For these and other details, the interested reader is referred to [11] for a complete specification of IPNL.

Sections 2 through 5 describe how IPNL implements the corresponding four major attributes of IPNL listed above. Section 2 describes IPNL's topology and addressing mechanisms, and lays down the basic architectural constructs. Subsequently, each section introduces additional key IPNL mechanisms; a basic understanding of IPNL is obtained only after reading all four sections. Section 6 describes our prototype implementation. Section 7 compares IPNL with other approaches, prominently IPv6, and Section 8 discusses next steps.

2. TOPOLOGY, ADDRESSING, AND ROUTING

The IPNL topology is the same as today's Internet topology: privately-addressed realms connected to the globally-addressed Internet, and, sometimes, to each other, by NAT boxes. The NAT boxes are called *nl-routers*, and the globally-addressed part of the Internet is called the *middle realm*. Privately addressed realms are called *private realms*.

An nl-router that connects a private realm with the middle realm is called a *frontdoor nl-router*, or simply a *frontdoor*. An nl-router that connects two private realms is called an *internal nl-router*. A single physical device can be both a frontdoor and an internal nl-router. These entities are shown in Figure 1.

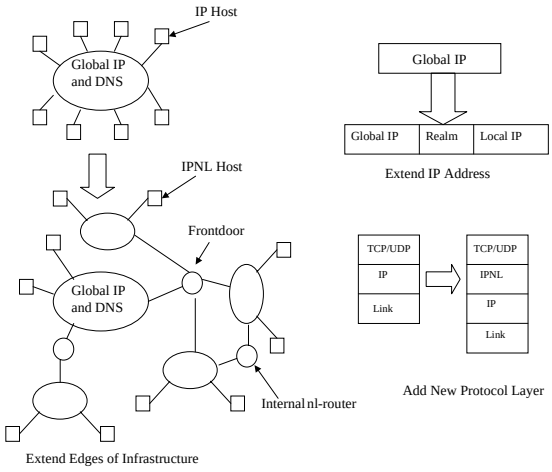

Figure 1: IPNL Topology

To IP routers[5] in a realm, an nl-router appears to be just another host. To nl-routers, a realm appears to be a multi-access non-broadcast "link". The "link-layer" protocol of this non-broadcast link is IPv4. In IPNL, the IPNL header is the end-to-end addressing header, and the IPv4 header is delegated the role of an encapsulating "link" header. In other words, at every nl-router hop, the IPv4 header of the incoming packet is stripped away, and a new IPv4 header is attached to the outgoing packet.

IP addresses for a given realm have no meaning outside that realm and never appear in IP headers outside of that realm. This is in contrast to today's situation, where a global IP address does have meaning in a private realm, but not vice versa[6]. This IP address

[5]Henceforth, IP boxes mean IPv4 boxes, and IP addresses mean IPv4 addresses.

[6]Of course, during transition, NAT and two-faced DNS will continue to operate as they do today, and, in this non-IPNL context, a global address does have meaning in a private realm.

isolation partly extends to DNS as well—while there is a single namespace, the DNS infrastructure itself operates independently in each realm, with no knowledge about other realms. This implies no new DNS resource record types are required.

IPNL headers can carry two kinds of routable addresses. One is the FQDN of the host, and the other is the IPNL address of the host. IPNL addresses are fixed-length numerical addresses. Datagram packets may be addressed using FQDNs only, IPNL addresses only, or both. Nl-routers can route packets using either type.

The FQDN serves as a somewhat static "long-term" address. While a host may have multiple FQDNs, the FQDN used for a given connection (or socket instantiation lifetime) must not change during the connection. Applications would normally use the FQDN to identify other hosts, and pass the FQDN to lower layers through the socket API. In such cases, the application is unaware of the IPNL addresses of hosts (including itself).

The IPNL address, on the other hand, is much more dynamic. A host may have multiple IPNL addresses, and these may change during a connection. The FQDN is the glue that binds these multiple IPNL addresses together. FQDNs are transmitted in the initial packet for a connection in each direction. Subsequent packets typically carry only IPNL addresses. IPNL uses both FQDNs and IPNL addresses because FQDN addresses, while fully routable by nl-routers, are of variable length, and expensive to route on. IPNL addresses are short fixed length fields, and, while transient, have the advantage of being efficiently routable. IPNL uses FQDNs to bootstrap and maintain the IPNL addresses.

2.1 Routing by FQDN

Every realm has associated with it one or more DNS zones. This is necessary in order for FQDNs to be routable addresses. Conversely, every DNS zone is associated with exactly one realm (although its parent zone may be spread over multiple realms). The realm associated with a given DNS zone is called the *home realm* of the zone. It is possible for a host from a given zone to be attached to a realm other than the home realm. Such a host is said to be a *visiting host*, and the realm where it is attached is called the *visited realm*. In Figure 2, the home realm for a.com is realm R1. Host y.a.com is a visiting host at realm R6.

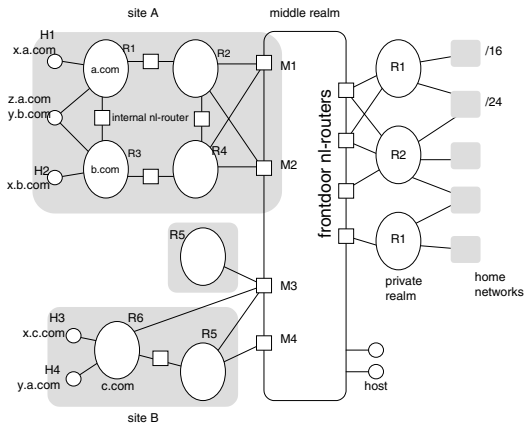

Figure 2: Example IPNL Configuration

We say that an internal nl-router is *behind* a frontdoor if it uses that frontdoor to reach the middle realm. Zone routing information is dynamically maintained in nl-routers so that a packet can be routed from any nl-router behind a given frontdoor to any zone[7] behind the same frontdoor. This routing information can consist of either an explicit forwarding table entry for the zone, or a default entry towards the frontdoor.

Typically, an nl-router would contain explicit routing table entries for zones in the same administrative domain, and a default entry would be used for all other zones. At a minimum, though, the frontdoor must have explicit routing table entries for all zones behind it. (An nl-router may also have explicit routing table entries for zones behind other frontdoors. These "backdoor" routes are not core to the routing architecture, however, and are not discussed further; [11] has the details for the interested reader.)

This zone routing information is established with dynamic routing algorithms. Zones are treated as maskable addresses in the same way that IP addresses are maskable. Whereas IP addresses are bit-maskable, zones are maskable only at the "dot" boundaries. Nevertheless, mechanistically, they are aggregatable in the same way that IP addresses are aggregatable. As such, multiple zones may be represented by a single routing table entry (for example, zones a.x.com, b.x.com, and c.x.com might appear as zone x.com in a routing table entry).

Zones are, of course, not as aggregatable as IP addresses, both because the assignment of domain names is primarily based on administrative closeness, not topological closeness, and because there are lots of administrative domains. Typically, we wouldn't expect to see much aggregation of zones across realms. Aggregatable zones would normally share the same home realm.

In practice, this lack of zone aggregatability is not a problem because nl-routers only need to keep explicit entries for a tiny fraction of all zones—namely those behind the same frontdoor. If a source and destination zone do not share the same frontdoor, packets are routed from the source zone to the frontdoor by default. The frontdoor then uses conventional global DNS to route[8] packets across the middle realm to the destination frontdoor.

Specifically, A-records in middle realm DNS refer not to the IP address of a host behind a frontdoor, but to the middle realm IP address of the frontdoor. Before a frontdoor can forward a received default-routed packet across the middle realm, it must first have done a DNS lookup over the middle realm to learn the IP address of the neighboring frontdoor.

Up to now, we have described how a packet is routed to a zone behind the same frontdoor, and how a packet is routed across the middle realm to a zone behind a different frontdoor. What remains to be described is how an internal nl-router forwards a packet to individual hosts attached to the same realm. For this, we require that internal nl-routers maintain the following per-host routing information:

- The FQDN and private realm IP address of all hosts in the realm, whether the host is visiting or not, and
- For each host whose home realm is the attached realm, but which is visiting another realm, the FQDN of a zone in the visited realm must be known.

An internal nl-router can learn of non-visiting hosts via a DNS zone transfer. Visiting hosts must register both with an nl-router in

[7] We say "route to a zone" as shorthand for "route to an nl-router attached to the home realm of the zone".

[8] This simply means that the frontdoor also acts as a DNS server, and does not mean that it queues up FQDN-attached packets waiting for resolution (see Section 3.3 for an example).

its home realm, and with an nl-router in its visited realm. When an nl-router receives such a registration, it, in turn, informs all other nl-routers attached to the realm. These neighbor nl-routers are learned through static configuration.

Because nl-routers must know about every host in its attached realms as well as about every other attached nl-router, it should be clear that private realms are not expected to be very big. They should have only a fraction of the over 16 million (Sections 2.2 gives the IPNL address format, including the sizes of various fields) possible hosts from the private address space.

To summarize, take the case where host x.a.com in Figure 2 is sending a packet to host x.c.com. Default routing gets the packet to frontdoor M1 (or M2). DNS information gets the packet from M1 to M4 (or M3). Dynamic routing on zones gets the packet from M4 to the R5-R6 internal nl-router. Internal nl-router R5-R6's host database gets the packet from there to host x.c.com.

2.2 Routing by IPNL Address

IPNL addresses are 10 bytes long, and consist of three parts (in order of high-order to low order):

1. A 4-byte globally unique IP address, which is the *Middle Realm IP* address (MRIP) of a frontdoor that the host currently uses to reach the middle realm.

2. A 2-byte *Realm Number* (RN) identifying the realm behind this frontdoor; because of the possibility of realm number translation (Section 3.2), the exact RN value in this field is meaningful only from the perspective of this frontdoor, and may differ from the RN value used by internal hosts within a site, and by other frontdoors.

3. A 4-byte IP address, which is the *End Host IP* (EHIP) address of the host within the realm specified by the RN field.

Neither RNs nor EHIPs are globally unique.

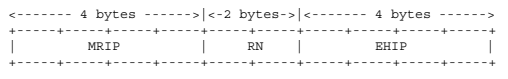

Figure 3: IPNL Address Format

In addition to being able to route to zones behind their frontdoors, internal nl-routers also know how to route to each realm using the 2-byte RN. This routing information is conveyed by the same dynamic routing protocol used for zones. Such a routing protocol would have several parallels with BGP, and, in fact, a modified BGP could be used. Whereas BGP calculates routes to Autonomous System (AS) numbers and associates IP prefixes with those ASs, IPNL's routing algorithm would calculate routes to RNs, and associate zones with those RNs. In addition, whereas BGP neighbors are reachable across ASs, nl-router neighbors are reachable across private realms.

Packets for realms behind different frontdoors are routed by default to the frontdoor. Frontdoors use the MRIP to forward packets across the middle realm. Once a packet reaches its destination private realm, the attached nl-router uses the EHIP to forward the packet across the private realm to the destination host. Note that the realm-routing protocol may establish different forward and reverse paths between a host and its frontdoor. Thus, we do not require any routing path symmetry assumptions[9].

[9] We, of course, assume that the destination uses the MRIP specified in the source address as part of the destination IPNL address for packets in the reverse direction.

Now, we repeat the example of a packet from host x.a.com to x.c.com, but using IPNL addresses instead. The destination address for the packet would be M4:R6:H3 (where M4 is the MRIP, R6 is the realm number, and H1 is the EHIP). Default routing gets the packet to M1 (or M2). MRIP M4 gets the packet from M1 to M4. Dynamic routing on RNs gets the packet from M4 to the R5-R6 internal nl-router. Internal nl-router R5-R6 uses the EHIP H3 to deliver the packet to host x.c.com.

2.3 Persistent Host Knowledge

IPNL hosts are configured with only two pieces of information: 1) their EHIP, and 2) their FQDN. Note that this is exactly the same information they are configured with for IPv4 today. In other words, no new configuration mechanisms (i.e., enhancements to DHCP) are required. Note too that an IPNL host does not keep persistent information about its MRIPs. Instead, these are learned dynamically, literally with every packet received (Section 4.1).

IPNL hosts must also learn the set of nl-routers attached to their realm. The basic approach is for the host to find one nl-router, using either IP anycast or a well-known domain name. This nl-router can then inform the host of the other nl-routers. Hosts periodically refresh this information.

2.4 The IPNL Header

Figure 4: IPNL Header Format

Figure 4 shows the IPNL Header. It consists of a 24-byte *local header*, an optional 16-byte *global header*, and an optional variable length *FQDN header*. The FQDN header contains, among other things, the source and destination FQDNs of the packet. There is no time-to-live (hop count) field. Nl-routers increment and copy the time-to-live field of the incoming tunnel's IP header to the outgoing tunnel's IP header. The IPNL header is designed so that the RN and EHIP of the source host are in the first 8 bytes of the packet. This allows nl-routers to identify the source host in received ICMPv4 messages. This is important for making debugging tools like traceroute and ping work across realms. Most of the other header fields will be described in subsequent sections.

2.5 Administrative Domains

There is no correspondence between administrative domains and IPNL topological entities. A large corporation may consume multiple frontdoors and all the realms behind those frontdoors (for in-

stance, the shaded area labeled "site A" in Figure 2). A small corporation may consume only a single realm, sharing an ISP-owned frontdoor with other corporations. A home network may consume part of a realm (for instance a /16 or /24) and share the realm with many other home networks. As such, an ISP could support over 17 million /16 home networks with a single global IP address (65536 RNs by 272 private /16's), for example.

3. SITE ADDRESS ISOLATION

This section describes the aspects of IPNL that allow site isolation. By site isolation, we mean that changes in any of the MRIPs of the site's frontdoors have no effect on internal packet exchanges or routing exchanges. In other words, the MRIP never shows up in any internal routing messages, host configuration messages, or headers of intra-site data packets.

A site's frontdoor MRIPs do, of course, show up in packets between internal and external hosts (called *global* packets). But because these MRIPs are learned on a connection by connection basis, there is no need to statically configure hosts and internal nl-routers with their site's MRIPs. This approach has two advantages:

1. There are no renumbering "events" where the addresses of all of a site's hosts must be simultaneously updated whenever the site's prefix changes. While infrequent, this prefix change entails massive changes to the entire site, and can be highly disruptive because all support infrastructure must be atomically updated.

2. Connections on a single host persist across multiple address changes on its interfaces. This feature is useful for mobile hosts that change realms or frontdoors. Given that private realms are envisioned to be small, this address change may be fairly frequent.

There are four primary mechanisms that provide site isolation:

1. Intra-site headers carry no MRIPs, and are not globally unique.

2. Realm number assignments are independently made, which allows sites that share the same frontdoor to independently number their realms.

3. IPNL addresses of packets are resolved in-flight, using FQDNs to route unresolved packets.

4. A 2-bit "Loc" field in global header (Figure 4) is used to indicate whether a global packet is behind the source or destination frontdoor.

These four mechanisms are discussed in turn.

3.1 Separate Local and Global Headers

When a host transmits an IPNL packet that does not cross the middle realm, it does not include the optional global header. Such packets are called *local* packets. Internal nl-routers receiving a packet without the global header know that the packet is to be routed locally. Therefore, there is no need for IPNL hosts to know their site's MRIPs. This also means that a site's internal traffic is not effected even if its MRIP changes.

Because locally addressed packets carry no MRIPs, they are not globally unique. Local packets behind other frontdoors can have the same headers. This use of a local packet is similar in many respects to the site-local address of IPv6 [14]. The IPv6 site-local address is distinguished by a specific 48-bit prefix, and like the IPNL local address is unique only within a site. The purpose of the site-local address in IPv6 is to lessen the impact of site renumbering. Thus, if an IPv6 site is renumbered, intra-site connections using the site-local address will not be affected by the change. By allowing hosts to dynamically learn MRIPs, IPNL takes this one step further, allowing even global connections to survive renumbering.

The use of locally unique addresses in both IPNL and IPv6 raises two problems:

1. How can the same local address from different sites be recognized as belonging to different sites, and

2. How does a host know that a given destination can be locally addressed?

IPv6 has no explicit mechanism to address the first question. It is up to site administrators to ensure that the context of all IPv6 site-local addresses is maintained. IPNL, on the other hand, uses FQDNs for this purpose. Because FQDNs carried in packet headers are fully routable, they can be used in lieu of IPNL addresses in configuration files. IPNL addresses may still have to be used directly when debugging network problems, so IPNL doesn't solve the problem completely. But such instances are clearly constrained and limited.

As for the second question, the primary mechanism by which IPv6 hosts determine whether a destination host is within a site or not is through the use of "two-faced" DNS. That is, DNS must know whether a given query is from a host within the site or external to the site, and compose its answer accordingly. It should be noted that the use of site-local addresses in IPv6 is entirely optional, whereas in IPNL it is mandatory. Therefore IPv6 users can get around the problems inherent in site-locals by simply not using them.

The way an IPNL host learns that a destination is locally addressable is through a technique whereby IPNL addresses are composed "in-flight" (Section 3.3).

3.2 Realm Number Independence

A problem comes up with realm number assignment when multiple sites share the same frontdoor. The purpose of site address isolation is to eliminate dependencies on the addresses assigned by ISPs, and by extension, from the addresses assigned to topologically nearby sites. Because all realm numbers behind a given MRIP must be unique, without some additional mechanism, all realm number assignments within sites that share the same frontdoor would have to be coordinated. This would defeat some of the benefits of site address isolation. For example, in Figure 2, the two sites sharing frontdoor M3 would have to coordinate their realm number assignments with each other. Without such coordination, both sites may assign the same RN numbers, as is shown with both sites having a realm numbered "R5".

IPNL allows independent realm number assignment by allowing different realm numbers to be used internal and external to a site. This is possible because realms are identifiable by their DNS zones. The neighbor nl-routers on either side of a site boundary know what realm number the other is using to identify a given realm. When the neighbors are internal nl-routers, this is scalably learned by the dynamic routing algorithm or by explicit configuration. When packets are locally addressed, the realm numbers are translated as the packet crosses site boundaries.

Because each frontdoor must have explicit routing information for all realms behind it, each frontdoor assigns realm number values to all realms behind it independent of the realm numbers assigned by the site. For globally addressed packets, the realm numbers assigned by the frontdoor are carried in the Global Dest Realm and Global Source Realm fields (Figure 4). These fields are tightly

coupled to the Dest MRIP and Source MRIP fields respectively. In other words, the realm number in the Global Dest Realm field is always the realm number assigned by the frontdoor defined by the Dest MRIP field. In the remainder of the paper, the Dest MRIP and Global Dest Realm fields are treated a single unit, and written as Dest MRIP+RN (and likewise for Source MRIP+RN).

For instance, in Figure 2, M3 recognizes that two realms have the same realm number, and externally represents one of the realms as having a different RN. As such, a host talking to a host on site B's R5 would see a different realm number, say R7. M3 would translate between the two RNs.

Every frontdoor can assign its own values for the external representation independent of other frontdoors. As such, it is a completely local and automatic function. For instance, M3 might represent site B's R5 as R7, while M4 represents it by R8. Different packets for a given connection may go through either frontdoor, for instance if one of the frontdoors crashes. As described in Section 4, IPNL has various mechanisms to maintain host identification in the face of such changes.

3.3 In-flight IPNL Address Resolution

When a host initiates a connection, it only knows three addresses:

1. its own FQDN,
2. its own EHIP, and
3. the FQDN of the destination.

In particular, it does not know its own MRIP and RN. It learns the destination host MRIPs by transmitting a message to a frontdoor asking that frontdoor to do a middle realm DNS lookup for the destination. But it does not know the destination's RN and EHIP when it transmits its first packet. Instead, the source MRIP and RN, as well as the destination RN and EHIP, are written into the packet as the packet travels from source to destination.

This is best described by example. In Figure 2, assume that host H1 is transmitting a packet to host H3. First, it requests a DNS lookup for H3's MRIPs from one of its own frontdoors. This is done by transmitting a request message hop-by-hop to each nl-router on the way to the frontdoor. The first nl-router that can answer the message does so. If no nl-routers can answer it, the frontdoor does a DNS lookup over the middle realm.

The initial packet from H1 to H3, then, contains both hosts' FQDNs, H1's EHIP, and one of H3's MRIPs (say M4). All other IPNL address fields are transmitted as values defined as "unknown". When the packet reaches the R1-R2 internal nl-router, it knows that the packet came from realm R1, and writes that RN into the Local Source Realm field. When the packet reaches frontdoor M1, it writes its MRIP and its representation of the source RN into the Used Source MRIP+RN fields (overwriting the Local Source Realm value, which is no longer needed once the packet traverses the frontdoor). When the packet reaches M4, it uses the FQDN to determine the RN for the destination zone. It writes its own MRIP and its representation of the destination RN into the Dest MRIP+RN fields. It also writes the local representation of the destination RN into the Local Dest Realm field. When the R5-R6 internal nl-router receives the packet, it looks up H3's EHIP and writes that into the destination EHIP field.

When H3 receives the packet, both IPNL addresses are complete. H3 stores the received values in a control block used only for this connection. In the return packet, the received Used Source MRIP+RN (M1+R1) are copied into the Dest MRIP+RN fields, the received Dest MRIP+RN fields (M4+R6) are copied into the Source MRIP+RN fields, and the FQDNs and EHIPs are reversed. When this packet exits site B, the exiting frontdoor writes it's MRIP+RN into the Used Source MRIP+RN fields. If the exiting frontdoor was M3, then H1 will receive a return packet with M1+R1 as the Dest MRIP+RN, M4+R6 as the Source MRIP+RN, and M3+R6 as the Used Source MRIP+RN.

H1 stores M4+R6 as a valid MRIP+RN for H3, but stores M3+R6 as the active MRIP+RN for H3. H1's return packet does not require the FQDNs. The Source MRIP+RN is copied from the received Dest MRIP+RN (M1+R1). This allows H3 to identify the source of the packet, even if, for instance, the packet exits site A through M2. The Dest MRIP+RN is set as M3+R6, and the EHIPs are reversed. Subsequent packets do not require FQDNs. The Source MRIP+RN is always used to identify the incoming packet. The most recently received Used Source MRIP+RN is always used to return packets.

Now consider the case of a packet from H1 to H2, a host in a different realm but behind the same frontdoor. In this case, when H1 transmits the DNS lookup request message, the first internal nl-router to receive the packet (say R1-R2) knows from the destination FQDN that the destination realm is local. The internal nl-router replies to this effect to H1. H1 then transmits a packet similar to the one in the previous example, but with the Dest MRIP field set to a well-known value meaning "none"[10]. The source RN and destination RN and EHIP fields are filled in as described in the previous example, but using the Local Source and Local Dest Realm fields. Once H1 learns these fields from the return packets, subsequent packets contain neither the FQDN nor global headers.

Consider a packet from H1 to z.a.com attached to the same realm. In the case the nl-router receiving the DNS lookup request message knows that the destination is in the same realm, and returns an answer to that effect including the destination host's EHIP. H1 then transmits a packet directly to z.a.com with the source and destination EHIP fields set, and the source and destination RN fields set to a well-known value meaning "this realm".

Finally, consider a packet from H2 to H4. H4's home realm is R1, but it is visiting realm R6. In this case, H2's initial packet will be routed to one of R1's nl-routers, say R1-R3. R1-R3 knows that H4 is in realm R6 with zone c.com, based on a registration message previously sent by H4 and distributed to nl-routers R1-R3 and R1-R2. R1-R3 will return a "redirect" message to H2 telling it the visited realm of H4. (As will be described in Section 5, the redirect is weakly authenticated by the Random ID (RID) field which makes spoofing hard, but is subject to man-in-the-middle attacks.) Subsequently, H2 will transmit a packet containing two Dest FQDNs in the FQDN header: y.a.com and c.com. FQDN c.com will get the packet routed to an nl-router of R6. This router will know the EHIP of y.a.com due to a previous registration from y.a.com, and the packet will be delivered.

3.4 The Location Field

The reader may have noticed a problem in the above H1-H3 example with regards to packets that do not contain the FQDN header. The problem arises because internal nl-routers do not know their own MRIPs. Without additional information in the packet header, the nl-router cannot know whether a packet should be routed to a local realm or to the frontdoor.

This problem is solved by the 2-bit location (or loc) field. The loc field is used only for packets that contain the global header but not the FQDN header. The loc field has three values: "behind the source frontdoor", "in the middle realm", and "behind the desti-

[10] IPNL header formatting requires that the global header be attached anytime the FQDN header is attached. In this example, however, the global header plays no role.

nation frontdoor". Packets are initially transmitted as "behind the source frontdoor". This tells internal nl-routers to default route the packet to the nearest frontdoor. This frontdoor changes the loc field to "in the middle realm" when it transmits the packet to the destination frontdoor. The destination frontdoor then changes the loc field to "behind the destination frontdoor" and forwards the packet towards the destination realm. Internal nl-routers behind the destination frontdoor continue to route the packet to the destination realm until it reaches the destination.

4. ROBUSTNESS

This section discusses robustness issues in IPNL, and describes the mechanisms used to achieve robustness. Specifically, we are concerned with how paths can be re-routed around failed nl-routers. Before discussing robustness in IPNL, we need to first consider what makes IPv4 robust. The primary principles of robustness in IPv4 are:

- Statelessness: There is no connection state in IPv4 routers. As long as an alternate path can be found around a failed router or link, communications between end-points can continue.
- Dynamic routing: This is what finds paths around failed routers.
- Neighbor pinging: Before dynamic routing can find alternate paths, a node (host or router) must first discover that its neighbor router is down. This is not possible unless either 1) each node has a relatively small number of neighbors (small fanout), or 2) there is native multicast between nodes in order to efficiently broadcast neighbor reachability information.
- Decoupled name resolution and routing.

IPNL attempts to follow these same principles. All nl-routers in IPNL are stateless (as long as both hosts are IPNL hosts. If one of the hosts is an IPv4-only host, then the nl-router must, of course, perform NAT, which is stateful). IPNL also uses a dynamic routing algorithm behind frontdoors. Although IPNL uses FQDNs as routable addresses, name resolution within a realm works the same way as it does today.

The only IPv4 robustness principle IPNL can not use is neighbor pinging between hosts and nl-routers across private realms, and between frontdoors across the middle realm. Both of these cases have a large fanout over non-multicast infrastructures. It is impossible for frontdoors to even know about each other, much less be expected to ping each other. Likewise, it is unrealistic to expect the nl-routers attached to a private realm to send reachability messages to all hosts attached to the realm.

IPNL uses two basic mechanisms to overcome this inability:

1. An "in-band trace" mechanism, and
2. Additional "path discovery" mechanisms in hosts, which are invoked after a timeout when no packets are received from a remote host.

4.1 In-band Trace

IPNL has two forms of in-band trace. One allows hosts to quickly detect failure of an nl-router in its realm. The other allows hosts to quickly detect failures in a destination host's frontdoor. The principle of in-band trace is best described through example. We start with the first because it is easier to understand.

For each connection, the host IPNL layer maintains a variable called "next-hop" that contains the IP address of the nl-router to which it should transmit packets for that connection. Next-hop is initially set to any attached nl-router. Subsequently, next-hop is always set to the source IP address of the latest received packet.

For example, assume that H1 is receiving packets from H4 via internal nl-router R1-R2. H1's next-hop value for this connection is R1-R2. Now, suppose that R1-R2 crashes. The dynamic routing algorithm operating in site A will discover this crash because of keep-alive messages between neighboring nl-routers across private realms. As a result, packets transmitted by H4 will be routed through nl-router R1-R3. When H1 receives such a packet, it will change its next-hop value from R1-R2 to R1-R3. Subsequent packets sent by H1 will go through R1-R3.

The principle here is that the presence of the source IP address in received messages represents a "trace" of part of the path from H4 to H1—specifically the last hop. This trace tells H1 how to route subsequent packets. The same principle applies to frontdoors.

For every globally addressed connection, the host IPNL layer maintains three pieces of information:

1. A list of MRIPs for the destination host (learned through DNS),
2. A list of MRIP+RN combinations for the destination host received in either the Used Source or Source MRIP+RN fields of previous packets for the connection, and
3. The latest Used Source MRIP+RN combination received.

Continuing the example of Section 3.3, assume that packets between H1 and H3 are using frontdoors M1 and M3 respectively. As long as this is the case, the Source MRIP+RN and Used Source MRIP+RN fields will match. Now assume that M1 crashes. Site A's internal dynamic routing will discover this. As a result, subsequent packets will exit site A through frontdoor M2. M2 will write M2+R1 into the Used Source MRIP+RN fields. When H3 receives this packet, it will recognize the source from the Source MRIP+RN fields (M1+R1), and also that a new frontdoor is being used (from M2+R1 in the Used Source MRIP+RN fields). Subsequent packets from H3 use M2+R1 in the Dest MRIP+RN fields, thus routing the packets through frontdoor M2.

Note that both of these trace mechanisms require that return packets be sent by the destination host. Fortunately, most applications send packets in both directions relatively frequently. However, some do not. In addition, the trace mechanism does not work for certain simultaneous failures. For instance, if both M1 and M3 in the previous example crash simultaneously, this mechanism does not work. Packets from H3 to H1 will indeed travel out of M4, but they will be transmitted to M1 and lost. H1, as a result, will never see a packet from H4, and will not learn that M3 has crashed. This leads to the need for the following path discovery mechanism.

4.2 Additional path discovery

If a host unexpectedly stops receiving packets for a given connection, it does not know if the reason is because the host has died, or some combination of failures is preventing packets from being received. When this happens, the host IPNL layer takes proactive steps to resolve the problem. The trigger for this can be either from the IPNL layer's own timeout mechanisms, or by request from the upper layer. The latter is preferred because the upper layer has a better idea as to whether it should be receiving packets.

The IPNL layer takes a series of steps to resolve the problem. First, it pings its own nl-router to make sure that it is alive. If not, it pings other nl-routers until it finds one. If this succeeds, then the host, in turn, sends pings to the destination host using all of the MRIP+RN combinations it has learned for the connection. If still

no return packets are received, the host tries the MRIPs it learned from DNS, or if it hasn't queried DNS, it does so.

If all of this fails, then the destination host is considered unreachable. The application or the IPNL layer may choose either to terminate the connection, or to wait for a while and try again.

5. FQDNS AS OVERLOADED ADDRESSES

The biggest departure from current IP architectures is IPNL's use of FQDNs for end-to-end routing and identification. Note that we say routing and identification, not simply identification. This is because, in a sense, IPNL can be viewed as routing on FQDNs across the middle realm.

To understand the pros and cons of using FQDNs as both a locator and an identifier, we must first examine the role of the IP address in locating and identifying the destination. The fact that IP addresses are "overloaded", that is that they serve to both locate and identify hosts, has been discussed in numerous articles ([3] [21]). The primary issue has been whether the location and identification functions should be combined in a single address, as with IP, or split.

The first author of this paper came out strongly for the split approach a decade ago in an early IPng proposal called Pip [10]. While he still believes in the principle, he also acknowledges that the mechanism used in Pip, a simple 64-bit flat identifier, was naively inadequate. This is because it would have been trivial to spoof a host and hijack its packets from anywhere in the network (not just from a man-in-the-middle position).

The only way to really effectively separate location from identification is to use an identifier that is both cryptographic and independent of any network layer addresses. Recent work proposed in IETF, particularly the Host Identity Payload (HIP) [17][11] but also Purpose Built Keys (PBK) [1], take this approach. Both of these have an anonymous mode in which a public key infrastructure or exchange of keys in advance is not necessary, making them appropriate for general use over the Internet. IPNL as of yet does not use a HIP approach, but would likely do so if the approach pans out.

The overloaded approach is a simple and elegant way to subvert hijacking of packets, at least where there is no man-in-the-middle attack. Because routing algorithms enforce delivery of packets to the destination address, and because router neighbor relationships are manually configured and therefore relatively hard to spoof (though by no means impossible), making the address also the identifier makes it very hard to hijack packets. This powerful feature is primarily what makes the overloaded approach attractive to its proponents.

The negative aspect of overloading the address is that the identification part becomes dependent on where a host is attached to the Internet. This results in the renumbering issue that has caused so much concern and added complexity (renumbering algorithms, site-local addresses) in IPv6. It is primarily this negative aspect that makes the overloaded approach unattractive to its opponents.

IPNL attempts to get around this impasse through the use of three components:

1. The FQDN, which is used primarily as an identifier, but often as a kind of locator too,

2. The IPNL address, which is used primarily as a locator, but sometimes also as a short-term identifier (i.e. when the Used fields contain a different value),

3. A Random ID (RID), which is used purely as a per-connection short-term anonymous identifier in order to prevent spoofing.

We concede that we are not happy about there being three components. We wish that one were enough, because that would be a lot simpler. The need for three is a reflection of the difficulty of getting both location independence and spoof-resistance out of a single address.

We start with the FQDN. We maintain that the FQDN as used in IPNL is, in essence, an overloaded address. That is, it both locates and identifies a host in the same sense that the original IPv4 address both located and identified a host. The original IPv4 address had the following overloading characteristics:

1. **Immediately returnable**: A non-spoofed source address in a received packet can be used to transmit a packet back to the source host.

2. **Non-hijackable**: A spoofed source address in a received packet does not cause a return packet to go to the source host.

3. **Long-term identifiable**: A non-spoofed address can be used at a much later time to send packets to the same host. This can be an address that was learned from a received source address or through some other means.

We can say the same three things about the FQDN as used in IPNL. This leads to the question: how can an FQDN have the same address overloading characteristics of the original IPv4, and yet not be subject to the renumbering problem? The answer lies in three important differences between the way DNS scales and the way IPv4 scales.

1. DNS derives its scalability from caching (especially NS-record caching), while IPv4 gets its scalability from aggregation. The chain of pointers in DNS start at a handful of root DNS servers. If there were no caching, and all DNS queries had to go through the root servers, DNS would obviously never scale.

2. DNS is dependent on IPv4, whereas IPv4 has no such dependencies. Because of IPv4, any two DNS servers anywhere in the world can be configured as neighbors.

3. A leaf DNS domain has only one parent domain. Put another way, DNS domains are all single-homed. Therefore, the multi-homing issue doesn't even come up in DNS.

These three things taken together result in the property that the name of a host has no dependence on where that host connects to the Internet. As it turns out, this does not quite solve all renumbering issues, though it comes close. The issue here is that of a multihomed network that gets Internet connectivity from two ISPs, but gets its DNS service from only one of the ISPs. (By virtue of the third DNS scalability property above, it has no choice but to gets its DNS service from one and only one ISP.)

Consider again our example of an ISP L changing the prefix of 25,000 home subscribers. Suppose that 15,000 of these are multihomed as described above, and that 1/2 of them get their DNS service from some other ISP M. To change these 7,500 subscribers' prefixes, for each subscriber ISP L has to communicate this change to ISP M. This would be far less difficult than getting subscribers to renumber their home networks as with IPv6. For instance, the ISP can discover which ISP is hosting its customers' DNS simply by doing a DNS query on the customers' domain name. Note that this same problem also exists for IPv6 (in addition to the renumbering issue).

[11] These are only Internet drafts as of yet, and so, strictly speaking, should not be cited.

There are some other important limitations regarding the use of FQDNs as overloaded addresses. The main limitation is that, to return a received packet, the returning host must do a DNS lookup in the return direction to prevent a spoofed source FQDN successfully hijacking a packet. A rogue host would have to hack into DNS to hijack packets. While this extra DNS lookup is an acceptable overhead for many cases, particularly peer-to-peer, it is no good for heavily loaded servers. The cost of doing the DNS lookup in the return direction for every received connection is too high for such servers, and could be used against the server in a Denial of Service (DoS) attack. Note that this applies only to packets that cross the middle realm. Behind a frontdoor, a traditional routing algorithm (FQDN-based) prevents hijacking in exactly the same way that IP routing algorithms do today.

Another limitation is that we cannot efficiently include a pair of FQDNs in every packet. The IPNL address, including the Used fields, and the RID are used to overcome these two weaknesses.

As described in Section 4.1, when an IPNL host talks to another IPNL host, its IPNL layer keeps an FQDN and a list of one or more IPNL addresses for the other host. Any of these (the FQDN and the IPNL addresses) can act as overloaded addresses once they are "securely" obtained[12].

The question is, given that an IPNL host has an FQDN that it trusts, how does it obtain IPNL addresses that it can also trust? The simplest answer is for the host to do a DNS lookup on the FQDN. This approach always applies to the initiating host. It also applies to respondent hosts that can afford the extra overhead.

The only case remaining is where the respondent host cannot afford the extra overhead of a DNS lookup. Specifically, this is the case of hosts anonymously accessing a heavily loaded server. By anonymous, we mean the case where the server does not care "who" is accessing it (or if it does, it resolves that at higher layers, for instance using a cookie or a login/password). The only thing it cares about, at least at the network layer, is that once it does identify a host, packets sent to the identified host indeed go to that host[13].

For these anonymous accesses, the IPNL address in the first packet received is, by definition, trusted. Even if the initiating host is lying about its FQDN, the respondent host doesn't care. Because it is an anonymous access, all the respondent host cares about is that packets returned to the initiating host get to the initiating host. Of course, if the respondent host is logging information about the access, it must log both the FQDN and the IPNL address as a unit. It cannot log only the FQDN and assume that the IPNL address is the right one for that FQDN.

Note that a host lying about its FQDN (for anonymous accesses) does not affect traffic for the host that genuinely owns that FQDN. This is because IPNL does not generalize about FQDNs not learned from DNS. It isolates all knowledge about FQDNs and IPNL addresses to the individual connections that produced that knowledge.

For example, say host X has address Xa and FQDN X.com. Another host Y has address Ya, and pretends to have FQDN X.com. Host Y anonymously accesses respondent host R with address Ya and FQDN X.com. R creates a record for this specific connection (i.e. the port numbers and protocol), and remembers that the host has FQDN X.com and address Ya. Next host X accesses respondent host R with address Xa and FQDN X.com. R simply creates another record for this specific connection and does not associate the two connections in any way. In other words, it never tries to send a packet for host X's connection to Ya, nor does it try to send a packet for host Y's connection to Xa.

The tricky part comes when packets to and from an initiating host need to go through a different frontdoor than the one learned in the initial packet. This must be done in an efficient and trusted way.

The efficiency comes from the use of the Used fields, as already described in Section 4. We mention it again here just to point out the architectural role that the IPNL address fields are playing when a packet with new Used fields arrives. Specifically, the Used fields play the role of the "true" source address (the one that can be used to return packets), and the source IPNL address plays the role of an identifier only. In other words, the source IPNL address tells the host who the packet came from, and the Used fields provide a new overloaded address that should be used subsequently.

The RID is used to prevent a rogue host from trivially spoofing the Used fields to hijack packets, and works as follows: The initiating host picks a different random value for the RID for each connection. All packets in either direction for that connection contain the RID. Because a hijacking host cannot easily guess the RID, the respondent host can trust that the IPNL learned from the Used fields is indeed from the true initiating host. Of course the RID does not protect against MITM attacks. For this, true security is needed (HIP or IPsec). The same can be said for IPv4 or IPv6. Neither of these are secure against a MITM attack in the absence of true security. Note that the RID can also be used for mobility while providing the same amount of hijack protection for anonymous access.

As already mentioned, we are not entirely happy with the fact that IPNL requires three different mechanisms to achieve the level of hijack resistance that IPv4 has. The whole thing has a certain cobbled together feel about it that smacks of lacking a strong architectural principle. The thing it lacks is that the architectural principle that it does have—an overloaded address (the FQDN) with scalability based on caching rather than aggregation—is not usable in the common case of a heavily loaded server.

A stronger architectural principle from which IPNL might benefit is a cryptographically strong host identifier. In HIP, the identifier is a 128-bit hash of a public key which is subsequently represented in packets as a 32-bit tag (specifically, the IPsec SPI [15]). If used in IPNL, the host identifier would take the place of the RID.

A cryptographic identifier could also potentially take the place of the Used fields. This is because the originator of the packet would be identifiable from the host identity alone, so the source MRIP could always be that of the frontdoor through which the packet actually passed.

HIP is still in the process of receiving community review, so it is premature to say whether it or something similar will suffice. HIP certainly has some cost associated with it—a Diffie-Hellman key exchange in a four packet handshake. It is possible that these costs are also prohibitive for the heavily loaded server case. Having said that, HIP has a cookie challenge mechanism that makes DoS attacks harder, so in certain ways it improves the situation for heavily loaded servers.

6. PERFORMANCE EVALUATION

We prototyped IPNL in the Linux 2.2.16 kernel by implementing the nl-router functionality in Click [16], and the host functionality in the Linux TCP/IP stack. We altered about 50 files, most of them under the `net/ipv4` kernel source subtree, to fully implement the host unicast algorithms, and added a new element directory called `ipnl` under the `elements` subtree to implement the internal and frontdoor nl-router algorithms. The prototype testbed consisted of

[12] We "secure" in quotes here because we don't mean strong security. There are no secret keys or encryption involved.

[13] We assume that if a respondent host does care about non-anonymous accesses at the network layer, it has the wherewithal to learn the MRIPs of the host in advance.

8 Linux boxes acting as end-hosts—4 each in 2 sites, with each site consisting of 2 realms, and another 8 Linux boxes acting as nl-routers, with an internal nl-router for each realm, and 2 frontdoor routers for each site. Each realm was configured with one DNS zone. We ran a simple BGP-like path vector protocol within each site to propagate both the realm reachability and FQDN routing information. The 2 frontdoor routers were to test the failover and load-balancing characteristics of the routing algorithms (the routing protocol sent traffic from each of the two internal nl-routers in each site to a different frontdoor under normal conditions, and to the remaining frontdoor under one frontdoor failure). Each of the two sites was interconnected by a network of 2 Bay networks Passport 8600 routing switches that forwarded the traffic between the 2 pairs of frontdoors. These routers simulated the middle realm, and could do only native IP forwarding.

We ran "netperf" [18] TCP throughput benchmarks to measure the overhead due to IPNL. The NIC cards used in the Linux boxes were 3Com 3c905B 10/100 Fast Ethernet cards, that could, without IPNL, attain a 99.1 Mbits/sec throughput for a packet size of 1500 bytes. After IPNL layer was added, we could see no degradation in the throughput at all. We also performed latency tests associated with a connection failover due to a frontdoor failure. Using a routing protocol that refreshed routes every 5 seconds, the end-to-end connection was restored after about 3 seconds on average. Using a link-state protocol to propagate the link-down failure event gave much better response times.

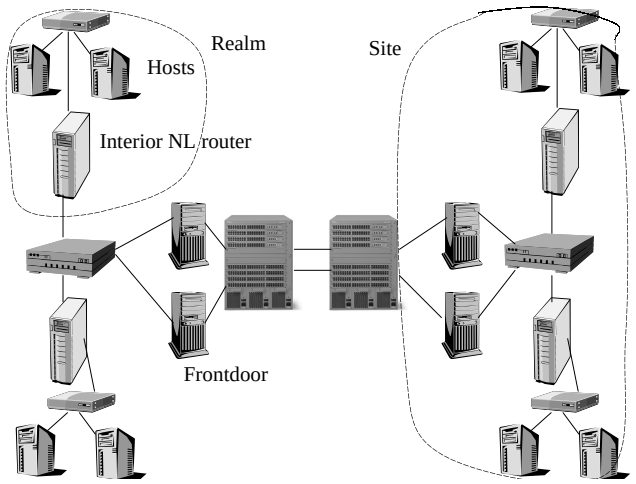

Figure 5: Layout of the Implementation Testbed

7. RELATED WORK

7.1 Comparison with IPv6

It is interesting to compare IPNL with various IPv6 transition mechanisms because their addressing and forwarding semantics share some commonality with IPNL. In one of the transitional deployments of IPv6, IPv6 is tunneled over IPv4. A deployment in which only the NAT boxes and hosts spoke IPv6 and always tunneled over IPv4 could be considered an extension of NAT. As it so happens, such an architecture is emerging from ngtrans, the IETF working group responsible for transition from IPv4 to IPv6.

There are two separate ngtrans working group projects contributing to this. Neither project explicitly sees itself as a NAT extension per se. This is purely our interpretation. As such, it should be understood that when we speak of a NAT box in what follows, we are referring to the IPv6 router that is positioned where a NAT box is normally positioned: between a site using private IPv4 addresses, and the globally routable IPv4 infrastructure. We call this NAT box a v6NAT box.

One of the projects, called 6to4 [2], uses the global IP address of the v6NAT box[14] as the prefix of the IPv6 addresses assigned to hosts behind the v6NAT box. When a packet with a 6to4 address arrives at a v6NAT box on its way out of a site, the v6NAT box reads the IPv4 address from the 6to4 prefix, writes it into the destination IPv4 address field of the tunneling IPv4 header, and transmits it across the globally routable IPv4 backbone.

The other project, called ISATAP [24], is much more recent. With ISATAP, the IPv4 address of the host is embedded in the lower 64 bits of the IPv6 address. As with 6to4, a v6NAT uses the embedded IPv4 address to automatically generate the appropriate tunneled IPv4 header In the case of ISATAP, however, the automatic tunnel is created by the v6NAT receiving a packet from the globally routable IPv4 backbone destined for a host behind the v6NAT box.

6to4 and ISATAP are unique among IPv6 transition tools in that they alter the semantics of the IPv6 address and the operation of the IPv6 forwarding engine! Without 6to4 or ISATAP, the IPv6 address is treated as a simple bit-wise best-match routing table lookup to determine the next hop IPv6 node. With 6to4 or ISATAP, a simple best-match routing table lookup is no longer enough to determine the next-hop: the router must additionally find a specific field in the IPv6 address and use that to determine the next hop. It is this change in the semantics of the IPv6 address to accommodate IPv4 that leads us to characterize 6to4+ISATAP as a NAT-extended architecture.

The primary objection to a v6NAT approach might be that it does not improve the scaling characteristics of the Internet, since IPv4 would continue to run as-is. While this is true, we point out that there is concern that even "native" (non-6to4) IPv6 will not improve on this situation even after it is widely deployed. This is because the renumbering requirements of IPv6 and complexities of multi-homing may result in individual site prefixes being advertised across the Internet core. Indeed there is enough concern here that the IETF has chartered a new working group specifically to look at these issues [22].

The primary difference between IPv6 tunneled over IPv4 and IPNL is that, even with 6to4+ISATAP, hosts must be aware of their address prefixes and must renumber when necessary. Another major difference is that once two IPv6 hosts start communicating with a given pair of addresses, they cannot change addresses, for instance, because of problems at the ISP connection point.

The GSE proposal of Mike O'Dell [7] proposed fully separating the identifier portion of the IPv6 address (the lower 64 bits) from the prefix (the upper 64 bits). The purpose of this was, among other things, to allow site multihoming by being able to change the prefix while still identifying the host. In this sense, GSE has parallels with IPNL.

The major problem with GSE is that the identifier could not be used to do DNS (or any other kind of) lookups to verify the mapping of prefix to identifier. Another problem was that GSE was tied to the IPv6 header, which does not include a Random ID field. As a result, GSE had unresolved hijacking problems.

[14] RFC 3056 itself does not refer to NAT boxes, but rather to "6to4 routers" that are "normally the border router between an IPv6 site and a wide-area IPv4 network". This is exactly where the NAT function resides.

7.2 Other Work

There are many recent efforts [4] [23] to provide enhanced services based on the notion of using FQDNs as persistent end-host identifiers.

TRIAD [4] is a recently proposed Internet architecture whose goal is to support an explicit content layer. Two of the major components of TRIAD are name-based routing (DRP), and wide-area relaying protocol (WRAP). These two components share two properties with IPNL. DRP uses FQDNs as the end-to-end identifier/address. WRAP, like IPNL, is a NAT-extended architecture. The primary difference between DRP and IPNL is that DRP proposes to globally distribute routes to DNS domains through a traditional routing protocol whereas IPNL uses DNS globally and traditional routing protocols only at the edges. Because of this, in spite of DNS aggregation, NBR is not likely to scale adequately—there are far more top-level domains (those under .com, .org, etc) than there are top-level IPv4 aggregations. Furthermore, WRAP uses stateful address translators that, while providing symmetric addressing[15], have the same well-known disadvantages of NATs—lack of robustness, scalability problems, and costly setup.

In [23], the authors use FQDNs as identifiers, and dynamically update DNS as part of an end-to-end approach towards supporting host mobility. Their focus is on providing a better mobility solution than Mobile IP [19]. While IPNL does not require any modifications to DNS to support mobility, their approach [23] relies on a securely-updateable DNS. The downside of not using secure DNS (or some other alternative such as a certificate infrastructure) is to introduce a certain amount of inelegance and uncertainty, as described in detail in Section 5.

8. NEXT STEPS

IPNL has a number of interesting and even promising characteristics, such as the various mechanisms for site isolation and scalable multihoming. It is premature, however, to say anything definitive about it. IPNL is a major new architecture, and it will take a lot of time, thought, and implementation experience from a broad community to be able to say convincingly that it works and works well.

Earlier in this paper, we argued that IPNL may be easier to deploy than IPv6 because it has two phases of deployment rather than three. This statement is only partially true. Before these three "box deployment" phases, there are at least three preliminary phases:

1. Standardization
2. Obtaining mindshare
3. Vendor implementation

In spite of the ongoing problems with IPv6, it has at least a 5 year head-start on the above three preliminary phases.

Given all this, an appropriate future for IPNL might be to pursue it as a community research project (perhaps under the auspices of the IRTF), and for IPv6 to incorporate whatever aspects of IPNL make sense.

ACKNOWLEDGMENTS

We thank Prof. Hari Balakrishnan for his valuable comments on earlier drafts of this paper. We also thank the anonymous reviewers for their useful comments and suggestions which helped improve the paper.

[15] This requires host protocol changes.

9. REFERENCES

[1] S. Bradner, A. Mankin, J. Schiller, " A Framework for Purpose Built Keys (PBK)", Internet Draft, draft-bradner-pbk-frame-00.txt, February 2001.

[2] B. Carpenter, K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC3056, February 2001.

[3] I. Castineyra, N. Chiappa, M. Steenstrup, "The Nimrod Routing Architecture", RFC1992, August 1996.

[4] D. R. Cheriton, M. Gritter, "TRIAD: A Scalable Deployable NAT-based Internet Architecture", Stanford Computer Science Technical Report, January 2000.

[5] M. Crawford, "Router Renumbering for IPv6", RFC2894, August 2000.

[6] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC2460, December 1998.

[7] Mike O'Dell, "GSE-an alternate addressing architecture for IPv6", Internet Draft, draft-ietf-ipngwg-gseaddr-00.txt, February 1997.

[8] R. Droms, "Dynamic Host Configuration Protocol", RFC1541, March 1997.

[9] K. Egevang, P. Francis, "The IP Network Address Translator (NAT)", RFC1631, May 1994.

[10] P. Francis, "Pip Near-term Architecture", RFC1621, May 1994.

[11] P. Francis, R. Gummadi, "IPNL Protocol Specification"; available from http://www.ipnl.net/spec/

[12] P. Gross, P. Almquist, "IESG Deliberations on Routing and Addressing", RFC1380, November 1992.

[13] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol", RFC2543, March 1999.

[14] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture", RFC2373, July 1998.

[15] S. Kent, R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC2406, November 1998.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The Click Modular Router", *ACM Transactions on Computer Systems*, 18(3), August 2000.

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

[18] http://www.netperf.org

[19] C. Perkins, Editor, "IP Mobility Support", RFC2002, October 1996.

[20] C. Rigney, A. Rubens, W. Simpson, S. Willens, "Remote Authentication Dial In User Service (RADIUS)", RFC2138, April 1997.

[21] J. Saltzer, "On the Naming and Binding of Network Destinations", RFC1498, August 1993.

[22] Site Multihoming in IPv6 (multi6), http://www.ietf.org/html.charters/multi6-charter.html

[23] A. Snoeren, H. Balakrishnan, "An End-to-End Approach to Host Mobility", *Proc. of 6th ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '00)*, August 2000.

[24] F. Templin, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", Internet Draft, draft-ietf-ngtrans-isatap-00.txt, March 2001.