Middleboxes No Longer Considered Harmful

Michael Walfish, Jeremy Stribling, Maxwell Krohn,
Hari Balakrishnan, Robert Morris, and Scott Shenker[*]
MIT Computer Science and Artificial Intelligence Laboratory
`http://nms.csail.mit.edu/doa`

Abstract

Intermediate network elements, such as network address translators (NATs), firewalls, and transparent caches are now commonplace. The usual reaction in the network architecture community to these so-called middleboxes is a combination of scorn (because they violate important architectural principles) and dismay (because these violations make the Internet less flexible). While we acknowledge these concerns, we also recognize that middleboxes have become an Internet fact of life for important reasons. To retain their functions while eliminating their dangerous side-effects, we propose an extension to the Internet architecture, called the *Delegation-Oriented Architecture* (DOA), that not only allows, but also facilitates, the deployment of middleboxes. DOA involves two relatively modest changes to the current architecture: (a) a set of references that are carried in packets and serve as persistent host identifiers and (b) a way to resolve these references to delegates chosen by the referenced host.

1 Introduction

The Internet's architecture is defined not just by a set of protocol specifications but also by a collection of general design guidelines. Among the architecture's original principles [12] are two tenets at the network layer (*i.e.*, IP layer) that are still widely valued, but are nonetheless often disobeyed in the current Internet:

#1: Every Internet entity has a unique network-layer identifier that allows others to reach it. During the Internet's youth, every network entity had a globally unique, fixed IP address. However, the emergence of private networks and host mobility, among other things, ended the halcyon days of unique identity and transparent reachability. Now, many Internet hosts have no globally unique handle that serves to direct packets to them.

#2: Network elements should not process packets that are not addressed to them. We call this tenet "network-level layering", and it implies that only a network element identified by an IP packet's destination field should inspect the packet's higher-layer fields. This decades-old guideline has become an empty commandment, as firewalls, network address translators (NATs), transparent caches, and other widely deployed network elements use higher-layer fields to perform their functions.

That these tenets are routinely violated is not merely an Internet legalism. The inability of hosts in private address realms to pass handles allowing other hosts to communicate with them has hindered or halted the spread of newer protocols, such as SIP [24] and various peer-to-peer systems [18]. Layer violations lead to rigidity in the network infrastructure, as the transgressing network elements may not accommodate new traffic classes. The hundreds of IETF proposals for working around problems introduced by NATs [54], firewalls, and other layer-violating boxes are compelling evidence that *middleboxes* (as such hosts are collectively known) and the Internet architecture are not in harmony [8]. Indeed, because middleboxes violate one or both tenets above, Internet architects have traditionally reacted to them with disdain and despair.

We take a different view. Rather than seeing middleboxes as a blight on the Internet architecture, we see the current Internet architecture as an impediment to middleboxes. We believe such *intermediaries*, as we will call them, exist for important and permanent reasons, and we think the future will have more, not fewer, of them.

The market will continue to demand intermediaries for various reasons. NATs maintain and bridge between different IP spaces.[1] Firewalls and other boxes that intercept unwanted packets will be increasingly needed as attacks on end-hosts grow in rate and severity. Since even sophisticated users have difficulty configuring PCs to be impervious to attack, we believe that users would want to outsource this protection to a professionally managed host—one not physically interposed in front of the user—that would vet incoming packets. Under the current architecture, such outsourcing to "off-path" hosts requires special-purpose machinery and extensive manual configuration. Intermediaries can also increase

[*]Affiliation: UC Berkeley and ICSI.

[1]Even if the move to IPv6 accelerates, some IPv4 networks will remain. Moreover, private address realms give some protection against certain types of network attacks. Hence, we do not think private IP spaces are a temporary inconvenience that will soon end.

performance through, for example, caching and load-balancing. Commercial service providers will continue to take advantage of such performance-enhancing intermediaries, disregarding architectural purity.

Thus, we have a fundamental conflict: although intermediaries offer clear advantages, they run afoul of the two tenets above, which causes harm and makes deploying and using intermediaries unnecessarily hard. Our goal, therefore, is an architecture hospitable to intermediaries, specifically one that allows intermediaries to abide by the two tenets, to avoid other architectural infractions, and to retain the same functions as today. Such an architecture would let a variety of middleboxes be deployed while also letting end-system protocols evolve independently and quickly.

Our architecture—which we call the *Delegation-Oriented Architecture* (DOA)—is based on two main ideas. First, all entities have a *globally unique identifier in a flat namespace*, and packets carry these identifiers. Second, DOA allows senders and receivers to express that one or more intermediaries should process packets en route to a destination. This *delegation* lets the resulting architecture embrace intermediaries as first-class citizens that are explicitly invoked and need not be physically interposed in front of the hosts they service. Globally unique identifiers and delegation have existed in previous work describing different architectures (*e.g.*, i3 [57]); see §9 for details. This paper's contribution is defining a relatively incremental extension to the Internet architecture, DOA, that coherently accommodates network-level intermediaries like NATs and firewalls. DOA requires no changes to IP or IP routers but does require changes to host and intermediary software. However, these changes are modular, and current applications can be easily ported.

We illustrate DOA with two examples: first, "network-extension boxes" which are analogous to today's NATs in their establishment of private addressing realms but do not obscure hosts' identities, and second, "network filtering boxes" which are analogous to today's firewalls but do not violate network-level layering and need not be topologically in front of the hosts they service. Our goal is to show how our architecture easily accommodates these boxes.

Scope and Limitations

DOA is based on a subset of the architecture in a previous paper [3]. That position paper touches on various issues—including mobility, multi-homing, network-level middleboxes, and application-level middleboxes—with scant attention to design details or implementations. In an attempt to bring some of those nebulous architectural mutterings into focus, this paper concentrates exclusively on network-level intermediaries and ignores their application-level counterparts.[2] This paper does not discuss mobility and multi-homing scenarios either (though DOA, because it separates location and identity, could—with modest extensions—handle those scenarios). Given our limited focus, DOA should be viewed not as a comprehensive architecture but rather as an architectural component designed to address network-layer middleboxes. Its design presumes IPv4 at the network layer but DOA is also compatible with, and useful for, IPv6.

The final limitation we mention is that some people want to deploy tenet-violating middleboxes (*e.g.*, a censorious government that silently filters packets entering and exiting national borders) and that DOA can neither prevent such architecturally suspect middleboxes nor mitigate their damage.

2 Background

This review of common network-layer middleboxes is limited to the two we build under DOA—NATs and firewalls—and to a subset of their drawbacks; for a complete review, see [8, 18, 23, 38, 55]. Although NAT and firewalling are often combined in one box, these two functions are logically separate.

2.1 NAT and NAPT

Network Address Translation (NAT) and Network Address Port Translation (NAPT) [54] have several uses. For the purposes of this paper, we assume the following common scenario: a NAT or NAPT box bridges two address realms, at least one of which is *private*. Private addresses are unique within an address realm but ambiguous between address realms [46]; public addresses are globally unique and reachable from all Internet hosts. The hosts in the private realm are said to be *behind* the box. Packets destined for hosts behind the box are said to be *inbound*. The difference between NAT and NAPT is that NATs do not look at fields beyond the IP header. We adopt the convention of referring to both NAT and NAPT as "NAT", though our description focuses on NAPT (the more common of the two today); for simplicity, we assume that NAPTs have only one external IP address.

People deploy NATs for two reasons:

- *Convenience and Flexibility:* Private addressing realms allow people to administer a set of hosts without having to obtain public IP addresses for each.
- *Security:* Since hosts behind the NAT do not have global identities, a host outside the private realm cannot address the hosts in the private realm or express that traffic should go to them, which protects them from unwanted traffic. Also, by default (*i.e.*, without manual configuration), a NAT allows only inbound

[2]The basic architectural ideas can be illustrated with network-level intermediaries. At the application level, one must consider how applications are structured and named, a topic outside this paper's scope [3].

traffic that is part of a connection initiated by a host behind the NAT.

The main operations performed by a NAT are: (1) dynamically allocating a source port at its public IP address when a host behind it initiates a TCP connection or sends a UDP packet; and (2) rewriting IP address and transport-layer port fields to demultiplex inbound packets to the hosts behind the NAT and to multiplex outbound packets over the same source IP address. NATs violate both tenets in §1. First, a NATed host's conception of its identity (namely its IP address) is a private address and thus is not a handle that it can pass around to allow other network entities to reach it. Second, NATs' modification of port fields violates tenet #2.

NATs cause the following additional problems:

- In order for a server behind a NAT to receive unsolicited inbound packets sent to a given destination port, one must statically configure the NAT with instructions about packets with that destination port. This manual step is called *hole punching* and requires administrative control over the NAT. The amount of manual configuration increases when a series of NATs separate a server from the public Internet, creating a tree of private address spaces[3]—in this case, one must not only configure each of the NATs in the tree but also coordinate among them; *e.g.*, each globally reachable Web server in a given tree of NATs must get traffic on a different port on the outermost NAT's public IP address. (By *outermost*, we mean "connected to the globally reachable Internet".)
- Hosts behind the same NAT cannot simultaneously receive traffic sent to the same TCP port number on the NAT's public IP address. However, some applications require traffic on a specific port; *e.g.*, IPSEC expects traffic on port 500 [44], so only one host within a tree of NATs can receive Virtual Private Network (VPN) [21] service.

2.2 Firewalls

A firewall blocks certain traffic classes on behalf of a host by examining IP-, transport-, and sometimes application-level fields and then applying a set of "firewall rules". It must be on the topological path between the host and the host's Internet provider, which we argue is unnecessarily restrictive. Today's firewalls disobey tenet #2 because, by design, they must inspect many non-IP fields in each packet. Since firewalls by default distrust that which they do not recognize, they may block novel but benign traffic, even if the intended recipient wants to allow the traffic.

[3]Such series of NATs are not artificial; see §5.4 and Figure 4.

3 Architectural Overview of DOA

This section gives an overview of DOA; we defer design details to §4. We first list desired architectural properties that aid in gracefully accommodating intermediaries and then describe mechanisms to achieve those properties. The remainder of the section discusses how DOA extends the Internet architecture.

3.1 Desired Architectural Properties

(1) Global identifiers in packets: Each packet should contain an identifier that unambiguously specifies the ultimate destination. The Internet architecture, as originally conceived, *did* provide global identifiers in packets, but IPv4 addresses no longer meet the "global identifier" requirement. (IPv6 addresses, because they reflect network topology, are also unsuitable for us, as we elaborate below.) The purpose of a global identifier is to uniquely identify the packet's ultimate destination to intermediaries in a way that is application-independent.

(2) Delegation as a primitive: Hosts should have an application-independent way to express to others that, to reach the host, packets should be sent to an intermediary or series of intermediaries. This primitive—called *delegation*—allows end-hosts or their administrators to explicitly invoke (and revoke) intermediaries. These intermediaries need not be "on the topological path".

3.2 Mechanisms

EIDs: To achieve property **(1)**, each host has an unambiguous endpoint identifier picked from a large namespace. Our design imposes the following additional requirements:

(a) The identifier is independent of network topology (ruling out IPv6 addresses and other identifiers with topology-dependent components, as in [42, 43]). With this requirement, hosts can change locations while keeping the same identifiers.

(b) The identifier can carry cryptographic meaning (ruling out human-friendly DNS names). We explain the purpose of this requirement later in this section.

To satisfy these requirements, we choose flat 160-bit endpoint identifiers (EIDs). A DOA header between the IP and TCP headers carries source and destination EIDs. Transport connections are bound to source and destination EIDs (instead of to source and destination IP addresses as in the status quo). DOA borrows the idea of topology-independent EIDs from previous work, including Nimrod [34], HIP [39], UIP [17].

EIDs are resolved . . .: DOA provides for delegation as a primitive by *resolving* EIDs. We presume a mapping service, accessible to Internet hosts, that maps

EIDs to some target specified by the EID owner. This resolution has two flavors:

- **. . . to IP addresses:** In order to communicate with an end-host identified by an EID, a prospective peer uses the mapping service to resolve the EID to an IP address. This indirection creates a way for a host to specify that prospective peers should direct their packets to a given delegate: the host has its EID resolve to the IP address of the delegate.
- **. . . to other EIDs:** More generally, an EID can resolve to another EID, allowing an end-host to map its EID to a delegate's *identity*; if an end-host's EID had to map to the delegate's *IP address* (or any other topology-dependent identifier), the end-host would have to update the mapping whenever the delegate's location changed. An EID can also resolve to a *sequence* of EIDs, each of which identifies an intermediary specified by the host. This sequence is carried in packets, yielding a loose source route in identifier space.[4] This option is reminiscent of i3's stacked identifiers.

Thus, our design requires an EID resolution infrastructure. We wish the management of this infrastructure to be as automated as possible, which is why we had requirement (b), above: automated management is easier if the EIDs are vested with cryptographic meaning [36]. The resolution infrastructure must scalably support a put()/get() interface over a large, sparse, and flat namespace. Distributed hash tables (DHTs) [2, 14, 49, 62] give exactly this capability, but any other technology that offers this capability would also suffice. DNS's "resolve-your-own-namespace" economic model cannot be used here, but there are plausible scenarios for the economic viability of a DHT-based resolution infrastructure [61].

We have not yet mentioned *sender*-invoked intermediaries. Under DOA, senders invoke intermediaries by putting into packets additional identifiers beyond the identifiers that resulted from resolving the receiver's EID. Sender-invoked intermediaries receive little attention in this paper but are part of DOA's design.

3.3 DOA and the Two Tenets

We elaborate on our earlier claim that DOA allows intermediaries to abide by the two tenets in §1. Because they are location-independent and drawn from a massive namespace, EIDs can globally and unambiguously identify hosts, satisfying tenet #1. As a result, a network element can reply to the source of a packet by sending to the location given by the resolution of the source EID.

To obey network-level layering (tenet #2), network elements need only follow normal IP layering rules, as follows. If an IP packet arrives at a network element and the packet's destination IP address is not the network element's, then the element may change nothing in the packet besides per-hop fields. (However, elements may drop packets based on information in the IP header, which permits functions such as ingress and egress filtering.) If, on the other hand, the packet's destination IP address matches the network element's, there are two cases: (1) The destination EID in the DOA header matches the network element's EID (*i.e.*, the packet has reached its destination); or (2) These EIDs do not match, which means the element is a delegate. In the latter case, network-level layering implies that the allowed packet operations are up to the entities in the delegation relationship.

Note that this last claim satisfies network-level layering but allows violations of higher-level equivalents, *e.g.*, an explicitly addressed firewall that looks at application payloads upholds the rules just given but flouts application-level layering. In general, this paper claims that DOA improves on the status quo by restoring network-level layering but does not insist that intermediaries adhere to higher-level layering. Why not? Higher-level layers define how to organize host software, and one can imagine splitting host software among boxes using exotic decompositions. Defining both higher-level layering and an architecture that respects these higher layers is a problem that requires care and one we have left to future work. In the meantime, we believe that hosts invoking intermediaries should decide how best to split functions between them and their intermediaries.

We now discuss how the IP layering rules given above apply to specific intermediaries. Under DOA, NATs, which exist to bridge address realms, need not obscure host identity: as we describe in more detail in §5, DOA-based NATs may rewrite IP fields but will neither touch DOA fields that carry host identities nor overload transport-layer fields. Also, firewalls could be explicitly invoked, meaning that packets ending up at the firewalls would be addressed to the firewall. While these new firewalls (which we cover in §6) could certainly have outmoded policies, causing them to drop novel traffic classes just as today's firewalls do, they are not violating network-level layering because packets are addressed to them. One result of this explicit addressing is that the firewall's invocation is under users' (or their administrators') control, so the user (or administrator) could decide to have packets destined for it sent to *another* firewall, one with better suited policies.

3.4 DOA and Internet Evolvability

The preceding point is more general than firewalls and is important for the Internet's flexibility and evolvability. Today, there is only one easy way to deploy a middlebox: putting it "on the path". Of course, under DOA, some

[4] In this case, transport connections are bound to the ultimate endpoint, which is identified by the last EID in the sequence.

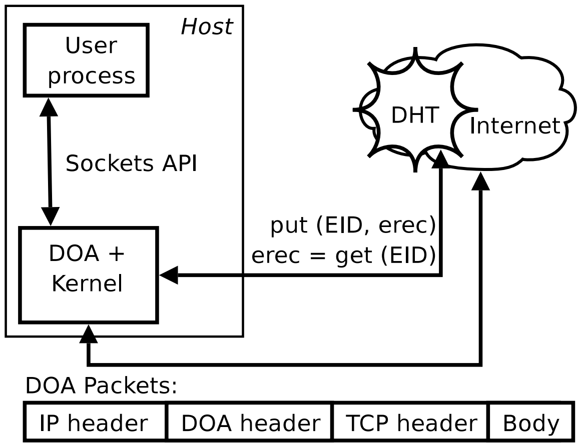

Figure 1: High-level view of DOA design.

boxes would have to be on the topological path to enforce physical security (*e.g.*, for denial-of-service protection); §6.4 describes how DOA accommodates these on-path boxes. However, DOA—with its flexible and application-independent invocation primitive—also gives users or their administrators the option to *outsource* functionality. Thus, under DOA, fewer intermediaries would need to be physically interposed, and users, no longer limited to the capabilities of the boxes in front of them, could avail themselves of a menu of services.

As a result, we believe that DOA could permit the rise of a competitive market in professionally managed intermediary services such as firewalls. Delegation and resolution are precisely what is necessary for such a market—the ability for users to select a provider and to switch providers. Because users would have a choice, they could seek the intermediary service that best suited their needs, and because these services would be professionally managed, they could keep up with the rapid pace of application innovation. Thus, we see DOA as contributing to the Internet's ability to evolve.

While we believe in its benefits, it is not clear that DOA is necessary for these new functions. In fact, we conjecture that even for those applications and intermediaries that one can seemingly build only under DOA, someone with enough imagination and fortitude could achieve equivalent functionality under the status quo—but not without running afoul of a basic tenet of the Internet architecture. We do suspect that the mechanisms of DOA will help new Internet functionality to evolve, but ultimately we believe our contribution is not novel *function* but rather novel *architecture*—making a class of network intermediary functions easier to build and reason about, and less likely to cause harm.

A natural question is how DOA relates to the canonical end-to-end argument [51], which is often interpreted as a warning against intermediaries. The central claim of the end-to-end argument is that application intelligence

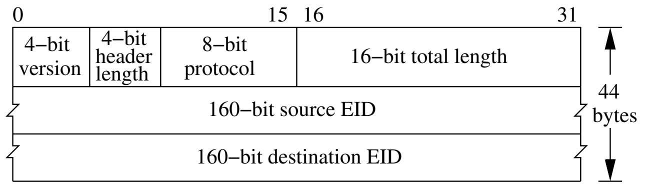

Figure 2: Example DOA header with no stacked identifiers.

is best implemented on end-hosts and not "in the network" because intelligence in the network leads to inflexibility and because end-hosts know best what functions they need. At a high level, DOA upholds this vision: the explicit invocation of intermediaries means that intelligence is not stuck in the network and that end-hosts can invoke the intermediaries that best serve them.

4 Detailed DOA Design

Given the preceding general description of DOA, we now present details of the design. Figure 1 shows the DOA components and the interfaces between them.

4.1 Header Format

DOA packets are delivered over IP, with the IP protocol field set to a well-known value. An example DOA header, with no extensions, is shown in Figure 2; the header length is measured in four-byte words, the protocol field is the transport-level protocol (*e.g.*, TCP, UDP) used by the packet, and the length field gives the DOA packet's total length (including the DOA header but not IP header) in bytes. TCP and UDP pseudo-checksums include the EIDs where IP addresses are used today (since transport logically occurs between two entities, each identified by an EID). The DOA header is extensible (*e.g.*, the remote packet filter presented in §6 extends the basic DOA header).

4.2 Resolution and Invoking Intermediaries

A DOA host wishing to send a packet to a recipient obtains the recipient's EID *e* out-of-band (*e.g.*, by resolving the recipient's DNS name to *e*). The sender then uses the EID resolution infrastructure—which is discussed in §3.2 and which we base on DHTs—to retrieve an `erecord`, depicted in Figure 3. An `erecord`'s fields are as follows: the EID field is the EID being resolved; the Target field is described in the next paragraph; the Hint field is optional information whose use we illustrate in §5; and the TTL field, like DNS's TTL, is a hint indicating how long entities should cache the `erecord`. DOA presumes that EID owners (or administrators acting on their behalf) maintain and possibly periodically refresh[5] the DHT's copy of their `erecord`.

[5]Some DHTs, like OpenDHT [29], store only soft state, requiring EID owners to do refreshes.

EID: `0x345ba4d...`
Target: EID$^+$ or IP address
Hint: *e.g.*, IP address
TTL: time-to-live, a caching hint

Figure 3: The `erecord`.

Recall from §3.2 that EIDs can either resolve to IP addresses (inducing what we call *EID-to-IP mappings*) or to one or more EIDs (inducing *EID-to-EID$^+$ mappings*). If the Target field of the `erecord` contains only an IP address i, then, as described in §3.2, the sender simply transmits a packet whose destination IP address is i and whose destination EID is e. In this case, the EID owner may or may not be directing potential senders to a delegate, but the semantics are the same: the EID owner is saying "to reach me, send your packet there".

If, on the other hand, the Target field of the `erecord` contains one or more EIDs, then the recipient is expressing its wish that the packet transit one or more intermediaries before reaching the recipient. In this case, the semantics are "to reach me, send your packet to these intermediaries in sequence". The sender would resolve the first EID in the series to an IP address j (perhaps via intermediate resolutions to other series of EIDs, each of which would be injected into the original series in the logical order) and send the packet to j. This stack of EIDs is carried in the DOA header; transport connections are bound to the last EID, which identifies the ultimate destination. (The design, but not our implementation, lets an EID resolve to multiple IP addresses; the multiplicity reflects a multi-homed host or an anycast situation in which a set of hosts are equivalent for the `erecord` owner's purposes. Similarly, each EID in the Target field could really be a *set* of EIDs, again representing equivalent hosts.)

To send a packet back to the source, the receiver executes the steps just described to resolve the sender's EID, f. The receiver cannot simply use the source IP address in the original packet as the destination IP address in the reply packet because f may resolve to a different IP address (*e.g.*, f's host sends packets directly but wants packets to it sent through an intermediary).

To spare the server the burden of a DHT lookup, the client can send its `erecord` as an optimization. (The client may have to send more than one `erecord` since the client's EID may resolve to a chain of EIDs before being resolved to the IP address needed by the server.) Also, DOA hosts use the `erecord`'s TTL to maintain a TTL-based cache of EID-to-IP and EID-to-EID$^+$ values, thus avoiding a DHT lookup for every packet.

The `erecord` and accompanying machinery exist to support receiver-invoked intermediaries. *Senders* invoke additional intermediaries by pushing the EIDs of the intermediaries onto an identifier stack in the DOA header.

4.3 Security and Integrity

Because identities (namely, EIDs) are separate from locations (namely, IP addresses), the following requirement arises under DOA: *The mapping from a given EID to its target must be correct,* i.e.*, either resolving an EID, or using an* `erecord` *directly sent by a host, must yield the IP address intended by the EID owner or by the EID owner's delegates.* Specifically, DOA must satisfy the following properties:

1. Anyone fetching an `erecord` must be able to verify that the EID owner created it.
2. Only the owner of an EID may update the corresponding `erecord` in the DHT.
3. When a host sends its `erecord` to another host without using the DHT, the sending host must not be able to forge the `erecord`.

To uphold these properties, DOA uses self-certification [36]: EIDs must be the hash of a public key, and the `erecord` is signed with the corresponding private key. When a host either performs a get() operation on the DHT, resulting in an `erecord`, or else receives an `erecord` directly from a purported EID owner, the host must check that the `erecord` is signed with the private key whose corresponding public key was hashed to create the EID in question. DHT nodes also perform this check before accepting `erecord`s. For more details, including how EID owners may update their public keys without changing their EIDs, see [61]; we adopt the mechanisms described there.

With the above properties satisfied, `erecord`s cannot be forged, but senders can still spoof source EIDs (*i.e.*, put the wrong source EID field in the packet). This attack is like spoofing a source IP address today (except that ingress and egress filtering, which help guard against IP address spoofing, are not applicable to EIDs): successful attacks do the same damage, and both attacks are detectable under two-way communication. For example, if a TCP client tries to spoof a source EID to a TCP server, when the server looks up the source EID (or uses the signed `erecord` supplied by the client), the server gets the *correct* (not fake) IP address for that EID, so when the server replies to the IP address, the host at that address will not complete the 3-way handshake.

Security of the DHT itself is a topic outside the scope of this paper. We briefly observe that DHT nodes cannot forge `erecord`s but can return old versions of `erecord`s. A way to guard against this attack by consulting multiple DHT nodes, instead of one, is mentioned in [14].

Also, we note that while IP source routing creates security problems, DOA's loose source routes of EIDs do not inherit these difficulties. With IP source routing, receivers reverse the source route to reply to a sender, which allows an adversary to carry out a man-in-the-middle attack by placing its IP address in a forged source route. Under DOA, however, hosts do not reverse the loose source route to reply to a sender.

4.4 Host Software

We now describe the software interface that a production DOA deployment would expose. Our prototype implementation differs from this description; see §7.1.

DOA software would run in the kernel and be exposed to applications with the Berkeley sockets API [37], which can extend to EID-based identification. Applications would open a new socket type, PF_DOA (in analogy with PF_INET, used by today's IPv4-based applications), and pass to the API a new data structure, the `sockaddr_eid`, which holds an EID and port (just as the `sockaddr_in`—which today's IP-based applications use—holds an IP address and port). Some of the socket calls, such as `connect`() and `sendto`(), might cause the DOA software, depending on the state of its caches, to issue one or more DHT lookups to resolve the EID into potentially intermediate EIDs and also an IP address. One could port today's applications by substituting `sockaddr_eid` for `sockaddr_in` in the code, though client applications would need additional logic to get a server's EID, perhaps via a DNS lookup.

For example, client TCP applications would call `connect`(), supplying a `sockaddr_eid` that contained an EID and port, both of which the application had obtained out of band. Similarly, TCP server applications would call `accept`(), getting back the EID and port of the initiating client. To reply to the client, the server's DOA software would resolve the client's EID to an IP address *i* and address reply packets to *i* at the IP layer.

For bootstrapping, DOA hosts would be configured with the EIDs and IP addresses of one or more of the DHT nodes, in analogy with how today's hosts learn the IP address of a DNS resolver (via hardcoding or DHCP). On boot up, the DOA software would insert into the DHT the host's `erecord` (which could contain an EID-to-EID$^+$ or EID-to-IP mapping, depending on the host's configuration) and would refresh the mapping periodically or in response to host configuration changes.

4.5 Limitations

DOA hosts cache `erecord`s, so hosts may have stale information about prospective peers. Also, two DOA peers in a TCP session resolve each other's EIDs only once—at the start of the session—so hosts cannot change locations without breaking existing connections. DOA could overcome this limitation if it were extended with a signaling mechanism, as in [39, 53], that allows hosts to notify peers of IP address changes. Finally, an EID owner cannot change which intermediaries are invoked based on who is trying to communicate with it.

5 Network Extension Boxes Under DOA

This section and the next describe example intermediaries under DOA. In the next section (§6), our focus is on filtering packets and how to move this function "off-path". In this section, we show how the DOA framework accommodates boxes that bridge between different IP address spaces and also simplifies the use of these boxes. Under the status quo, these boxes are known as NATs but would be reincarnated under DOA as tenet-upholding Network Extension Boxes (NEBs).

We first consider three usage scenarios for NEBs (§5.1), then give our general approach, including a short discussion of architectural coherence (§5.2), and then discuss the benefits of this approach (§5.3). One of the benefits, automatically exposing hosts behind NEBs, is particularly useful when NEBs are cascaded (§5.4). We present several mechanisms for achieving automatic configuration (§5.5) and require that they work when there are multiple levels of NEB. We conclude the section with a discussion (§5.6).

5.1 Scope

The following NEB scenarios reflect reasons for deploying NATs today (§2.1) and are ordered by the degree of access control:

(a) A host behind the NEB is accessible on all ports. The NEB creates a separate addressing realm but does not control access. Under this scenario, which corresponds to the "Convenience and Flexibility" reason for deploying a NAT today (§2.1), many hosts within an organization can be reachable as first-class members of the Internet, even if the organization has only one IP address.

(b) A host behind the NEB is accessible on configured ports, and the NEB blocks unsolicited traffic to the host on the other ports. This scenario, which reflects both reasons for deploying a NAT (§2.1), is analogous to, *e.g.*, today setting up a Web server behind a NAT and configuring the NAT to send all packets with destination port 80 to the Web server.

(c) A host behind the NEB is accessible on no ports, *i.e.*, the host can only receive packets associated with connections it has initiated. This scenario, which is principally driven by the "Security" reason for deploying a NAT (§2.1), is the default under NAT today.

We expect that under DOA, scenario (b)—a mix of access control and exposure—would be most common. However, for clarity, we focus on scenario (a) and return to scenarios (b) and (c) at the end of the section (§5.6).

5.2 Approach

NEBs preserve packets' DOA headers and use the destination EID field as a demultiplexing token. For example, the NEB could maintain an EID-to-IP table, look up the destination EIDs of incoming packets, and then use the results of these lookups to rewrite the destination IP addresses. There are other ways to demultiplex; we cover them in §5.5.

This approach upholds the two tenets stated earlier. Tenet #1 holds because an end-host behind a NEB can pass its EID to others, who can then use this handle to direct packets to the given host. As mentioned in §3.3, to obey network-level layering (tenet #2) NEBs may only rewrite fields in a packet if the packet is addressed to the NEB. Since NEBs, like today's NATs, have to rewrite both the destination IP addresses of inbound packets (to demultiplex them) and the source IP addresses of outbound packets (to make them appear as if they originated at the NEB), the discussion in §3.3 implies that both inbound *and* outbound packets be addressed to the NEB at the IP layer.

However, this approach, in pure form, makes the NEB resolve the destination EIDs of outbound packets. As a practical matter, sources of outbound packets could do the resolution and put the resulting IP address somewhere in the packet, thereby sparing the NEB this resolution burden. The source could even put the resulting IP address in the destination IP address field; at the IP layer, then, outbound packets would look alike under NEB and NAT. This modified approach—which technically violates the rules in §3.3 but is consistent with the spirit of the tenets because the violation is under the control of the end-host—is what we adopt.

5.3 Benefits

Upholding the two tenets results in the following benefits, some of which solve the problems stated in §2.1.

End-to-end communication. Communication is logically between two EIDs. Thus, protocols can uniquely identify hosts.

Ports are not overloaded. Not using the destination port as a demultiplexing token lets multiple hosts behind a NEB receive packets sent to the same destination port.

VPNs. Getting VPNs to work through NATs is cumbersome and complicated [44]. The difficulties under the status quo result from NATs rewriting both ports and IP addresses. Under DOA, NEBs do not rewrite ports, and the state associated with encrypted tunnels could be bound to EIDs, not IP addresses.[6]

Automatic configuration. Under DOA, the process of exposing a host behind a NEB can be automated. When NEBs are cascaded, a scenario covered in the next section, this automation is particularly useful—and particularly problematic under the status quo (§2.1).

[6]Much of the HIP work [40] focuses on such binding of IPSEC state to cryptographically imbued EIDs.

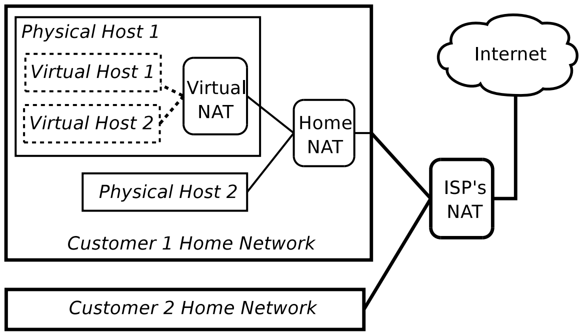

Figure 4: A tree of NATs.

5.4 Cascaded NEBs

The scenario of multiple address realms between a given host and the rest of the Internet is becoming more frequent. Consider the following example, depicted in Figure 4: an individual runs a virtual host (using, *e.g.*, VMWare [60]) that runs behind a NAT on the physical host (such NATing of virtual hosts is common). The physical host is in turn a member of a home network that is all behind a single NAT, which is connected to a broadband provider. The link from the broadband provider, owing to the provider's operations, is itself NATed, making, altogether, three levels of NAT between the virtual host and the global Internet.

We now cover protocols for automatically configuring NEBs to expose servers; we require the protocols to work when servers are behind multiple levels of NEB.

5.5 Secure Automatic Configuration

A protocol for configuring NEBs to expose servers must satisfy three requirements. First, the protocol must tell the end-host what to put in its `erecord` since an end-host separated from the global Internet by levels of NEB has no *a priori* knowledge about the IP addresses of NEBs between that end-host and the Internet. Second, the protocol must establish state, either in NEBs or in the EID resolution infrastructure, that allows NEBs to use the destination EID field in packets as a demultiplexing token for rewriting the destination IP address field.

Third, this state must correspond to the wishes of the actual EID owner, rather than of an impostor trying to divert the EID owner's traffic. This focus on authenticity is warranted because passing unprotected protocol messages through levels of NEB could be problematic. For example, an upstream provider cannot trust NEBs administered by its customers, and end-users cannot trust each other's NEBs to correctly propagate control or data messages. Also, NEB networks, like today's NATs, would often be constructed over wireless links, which are susceptible to eavesdropping and tampering. In what follows, we assume that a NEB trusts only the NEB directly upstream of it (called its *parent*); that NEBs and end-hosts know the EID of their parent; and that all links in the NEB network are vulnerable to eavesdropping, tampering, and arbitrary data injection.

We now give three mechanisms, each using a different kind of EID resolution, that meet the requirements above. We implemented the third one; see §7.2.

5.5.1 EID maps to EID

Each NEB and end-host creates a mapping in the global EID resolution infrastructure from its EID to its parent's EID; in other words, NEBs and end-hosts use the delegation primitive to say, "to reach me, send your packet to my parent's EID". Also, each NEB holds a mapping from its children's EIDs to its children's internal IP addresses.

Control plane. Assume an end-host with EID e_0 must traverse NEBs with EIDs e_1 through e_n before reaching the Internet. The end-host inserts a mapping from its EID (e_0) to its parent's EID (e_1) into the global EID resolution service. The end-host also sends a message to e_1 informing it of a mapping between its EID (e_0) and its IP address (i_0). All other internal NEBs in the chain (e_1 through e_{n-1}) use the same protocol. The outermost NEB uses the global EID resolution infrastructure to map its EID (e_n) to its IP address (i_n), which is globally reachable. A NEB with EID e_{j+1} should only accept an EID-to-IP mapping of the form $\langle e_j, i_j \rangle$ if the mapping is authentic, *i.e.*, if it is signed by the private key corresponding to e_j; performing this check might require e_j to send e_{j+1} its public key (which should hash to e_j).

This approach, as just described, is vulnerable to replays of $\langle e_j, i_j \rangle$ mappings. Such replays would allow the wrong end-host—one that is later assigned IP address i_j—to redirect e_j's traffic to it. We show how one might protect against these attacks in §5.5.3.

Data plane. Assuming the end-host and intermediate NEBs all initialize successfully, a remote client can send data packets to the end-host (with EID e_0) by using the EID resolution infrastructure to map e_0 to e_1, e_1 to e_2, and so on, up the NEB chain. The last EID lookup maps e_n to the IP address i_n. The client then stacks the identifiers e_0 through e_n in its packets and sends the packet to IP address i_n. Once the packet reaches the outermost NEB (e_n), the NEB pops the top EID off the stack to find that e_{n-1} is the packet's next hop. The NEB then consults its routing table to map EID e_{n-1} to IP address i_{n-1}, rewrites the packet's destination IP address to i_{n-1}, and forwards the packet. This process continues until the packet reaches its eventual destination, e_0.

5.5.2 EID maps to EID and a Hint

Another approach uses the `erecord`'s Hint field, mentioned in §4.2, to relieve NEBs of state.

Control Plane. The end-host inserts into the EID resolution infrastructure a mapping from its EID, e_0, to the EID, e_1, of its parent NEB; the `erecord` holding this

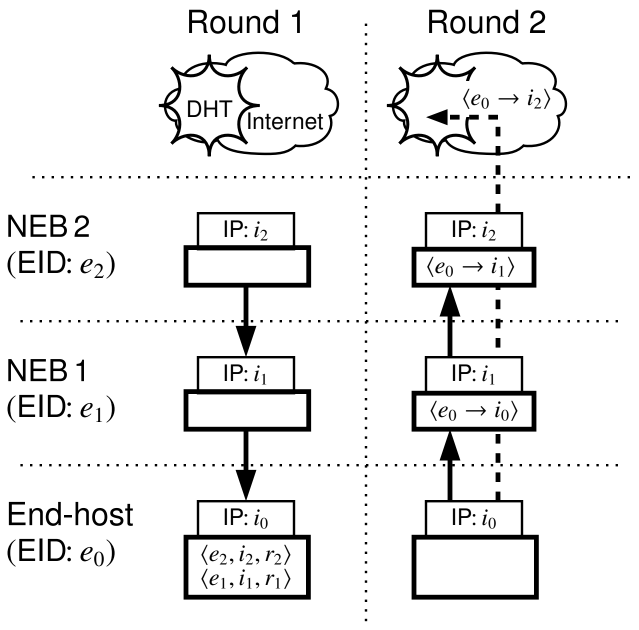
Round 1
Round 2
DHT
Internet
$\langle e_0 \rightarrow i_2 \rangle$
NEB 2
(EID: e_2)
IP: i_2
IP: i_2
$\langle e_0 \rightarrow i_1 \rangle$
NEB 1
(EID: e_1)
IP: i_1
IP: i_1
$\langle e_0 \rightarrow i_0 \rangle$
End-host
(EID: e_0)
IP: i_0
$\langle e_2, i_2, r_2 \rangle$
$\langle e_1, i_1, r_1 \rangle$
IP: i_0

Figure 5: NEB and DHT state after each DOA-RIP round.

mapping has in its Hint field the end-host's internal IP address, i_0. The NEB e_1 likewise creates a mapping in the EID resolution infrastructure from its own EID to the EID, e_2, of its parent and puts its "outer" IP address, i_1, into the Hint field of the `erecord`. This process continues until the outermost NEB inserts a mapping from its EID, e_n, to its "outer" IP address, i_n.

Data plane. A remote host wishing to communicate with e_0 resolves e_0 to e_1, e_1 to $e_2, \ldots, e_{n-1}$ to e_n, while remembering the Hints $i_0, i_i, \ldots, i_n$. As with the previous mechanism, the remote host stacks the identifiers e_0 through e_n in its packets—and in this case also includes in the DOA header the IP addresses i_0 through i_n—then sends the packet to IP address i_n. Once the packet reaches the outermost NEB (e_n), the NEB pops the top EID and IP address off the stack to find that e_{n-1} with IP address i_{n-1} is the packet's next hop, and the process continues.

5.5.3 EID maps to IP address

The previous two mechanisms require a prospective sender to do as many EID resolution infrastructure lookups as there are levels of NEB. An alternative, that we call DOA-RIP, allows senders to do a single resolution: from the EID, e_0, of the end-host to the IP address, i_n, of the outermost NEB.

Control plane. End-hosts and NEBs follow a two-round protocol, depicted in Figure 5. In the first round, the end-host (with EID e_0) sends an initialization message to its parent in the NEB tree; intermediate NEBs forward the message until it reaches the outermost NEB (with EID e_n). The outermost NEB creates a message

$x_n = \langle e_n, i_n, r_n \rangle$ (r_n is a random nonce to prevent replay attacks), signs x_n, and sends it to the NEB with EID e_{n-1}. Each NEB e_k ($k < n$), follows suit, appending the message $x_k = \langle e_k, i_k, r_k \rangle$ to x_{k+1}. When the end-host receives x_1, it verifies the message using e_1's public key. This message is a *route* to the global Internet.

In the second round, the end-host creates a series of requests $y_k = \langle e_0, i_{k-1}, r_k \rangle$ for $1 \le k \le n$; signs each y_k individually; concatenates all the y_k's and appends its public key; and sends this message up the NEB chain. Each NEB e_k verifies y_k using e_0's public key and signature. Each NEB further checks that r_k is in its cache and that r_k is the nonce it issued in the first round for EID e_0 (the NEB flushes r_k from a cache within a fixed number of seconds—10, in our implementation—of issuing r_k). If these checks succeed, the NEB flushes r_k, establishes a mapping $\langle e_0, i_{k-1} \rangle$, and propagates the request up the NEB tree. If all NEBs successfully establish the mapping, the end-host inserts into the EID resolution infrastructure a map from e_0 to i_n.

Data plane. To communicate with the end-host, remote clients first resolve e_0 to i_n and then send packets with destination IP address i_n and destination EID e_0, at which point the outermost NEB, and all succeeding NEBs in the chain, use their internal state to forward the packet to the end-host.

5.6 Discussion

Other scenarios. Though we focused on scenario (a) (from §5.1), the benefits noted above (in §5.3) apply equally to scenario (b). Two of the three mechanisms for automatic configuration also apply (the stateless NEB from §5.5.2 does not) with the one change that end-hosts—when making signed requests of parent or ancestor NEBs to add EID-to-IP mappings—need to add requests to open (or block) specific ports. This type of automatic hole punching works under DOA, in contrast to the status quo, for three reasons: (1) DOA has a persistent notion of host identity, which allows NEBs to associate policies with hosts and remote network entities to identify hosts behind the NEB; (2) port fields are not overloaded under DOA, so internal nodes in the NEB tree do not have to coordinate among themselves, in contrast to the status quo wherein only one server in a tree of NATs can receive, *e.g.*, traffic destined to port 80 on the outermost NAT's public IP address; and (3) hosts can leverage the cryptographic properties of their identities to create signed messages saying "handle my packets like this".

The benefits above, except automatic configuration, also apply to scenario (c). Although this scenario is the strictest access control NEBs offer, network administrators may still prefer NATs, since NATs, unlike NEBs, obscure the identities of the organizations' end-hosts. Our response is that organizations today use NATs in part because they hide internal network topology. Since EIDs are independent of network internals, organizations might be looser about exposing EIDs than IP addresses.

Comparison of the mechanisms. Observe that the three mechanisms above are different ways to perform routing that offer different trade-offs between state held in the NEB and the degree of fate-sharing. With one of the mechanisms (§5.5.2), all information about EID-to-IP mappings is in the EID resolution infrastructure, which simultaneously frees the NEB of state but makes correct routing depend on the availability of the resolution infrastructure. In contrast, DOA-RIP pushes nearly all state into the NEBs along the path between two communicating entities.

6 Network Filtering Boxes Under DOA

In this section, we demonstrate DOA's delegation primitive with a simple remote packet filter (RPF) box that yields functionality similar to today's firewalls but need not be interposed between a host receiving firewall service and that host's link to the Internet. One can certainly get similar functionality today with special-purpose machinery (*e.g.*, VPN software, though their interfaces differ across solution providers). However, we believe that decoupling services from topology is best done with *architectural*, rather than *application*, support because: (1) users should be able to compose intermediaries and (2) users should be able to change their delegates easily (see §3.4), both of which imply that the architecture support a standard, application-independent invocation method.

6.1 Approach and Design

The RPF is a basic application of DOA's mechanisms; it is depicted in Figure 6. A user (or representative of the user, *e.g.*, corporate IT staff) wanting remote firewall service creates a mapping in the EID resolution infrastructure from the end-host's EID, e, to the RPF's EID, f (or to the RPF's IP address, but then if that IP address changes, the resolution of e will be incorrect). This end-host expresses its actual network location either by putting its IP address, i, in the Hint field of the `erecord` to which e resolves, or by communicating directly with the RPF and telling it about the association between e and i. (Our implementation, described in §7.3, uses the second option.)

When a sender attempts to contact e, it first looks up e in the EID resolution infrastructure, sees that e maps to f, and then further resolves f to an IP address (which might involve intermediate resolution steps, depending on whether the RPF itself has delegates). In the simple case in which f resolves directly to an IP address j, the sender forms IP packets with destination address j and destination EID e. Note that f must be in the *stack* of identifiers since the host given by j may actually be the RPF's *delegate* rather than the RPF itself (*e.g.*, if the RPF

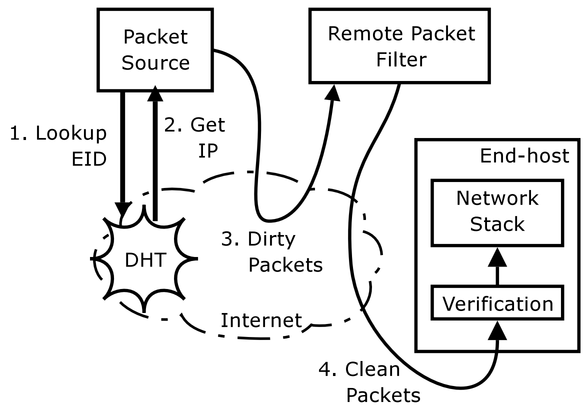

Figure 6: Packet filtering under DOA using delegation. End-hosts apply a simple verification rule, not a collection of them.

were behind a NEB, the NEB would need *f*'s EID to make a decision about the packet).

When receiving IP packets, the RPF extracts the destination EID *e* from the DOA header, looks up the set of rules associated with *e*, and finally applies these rules; examples of such rules are filters to block or accept traffic based on IP- or transport-layer fields. The result is "passing" or "failing" a packet. When packets "fail", the RPF drops the packet.

The RPF attests that packets "passed" by inserting into the packet a MAC (Message Authentication Code) taken over the packet; the MAC is keyed with a secret shared between RPF and end-host. The RPF then rewrites the packet's destination IP address and sends the packet to the end-host, which applies a single rule: redoing the MAC computation and testing whether the result matches the MAC in the packet. The end-host ignores packets that fail this test; thus, only packets that have been vetted by the RPF are processed by the end-host's networking and application software. The MAC is carried in a DOA security header, which extends the standard DOA header described in §4.1.

The RPF depends on both of DOA's core mechanisms: first, because of unique host identifiers, the RPF has a way (namely the destination EID field) to distinguish among hosts, allowing it to apply host-specific rules and then send the processed packet to the correct destination. Second, the delegation primitive is what allows the RPF to be invoked in the first place. See §7.3 and §8.3 for implementation and evaluation details.

6.2 Benefits

We first claim two architectural benefits, as discussed in §3.3 and §3.4: the RPF described here does not violate network-level layering, and also, a market for such services could arise.

These architectural benefits lead to simplification for users. Getting firewall rules right is hard, far beyond ordinary users' ability, and commercial products (*e.g.*, Norton [59]) require users to keep their software current. Outsourcing per-packet rules to a central provider solves those problems. Of course, end-hosts still have to check packets, but the check—"was this packet vetted by my RPF provider?"—is considerably simpler than the usual complement of firewall rules.

6.3 Limitations

The box just described is primitive. For it to provide the same functions as today's firewalls—such as using existing TCP connections, and not just stateless filtering rules, to make decisions—protected end-hosts would have to direct their outbound traffic through the RPF. These end-hosts would use the mechanism of sender-invoked intermediation (§4.2).

6.4 Physical Security

Some organizations require that every inbound and outbound packet be vetted by a box that is physically interposed between the organization and its link to the Internet. We briefly describe two scenarios for such on-path boxes under DOA.

We start with an on-path vetter that works with an off-path RPF. As above, an end-host within the organization, *h*, creates a mapping in the EID resolution infrastructure from its EID, *e*, to the RPF's EID. In this case, however, *h* tells the RPF that after the RPF processes packets destined to *e*, it should send them to the vetter's IP address (instead of to *h*'s IP address, as above). The vetter allows packets into the organization only if they are addressed to it at the IP layer and if the MAC check succeeds, thereby ensuring that the RPF has checked every packet entering the organization. The vetter uses the destination EID field to forward vetted packets to the correct host.

Some organizations will of course not want an RPF, preferring to deploy an on-path firewall and manage the rules itself. DOA supports an on-path firewall just as it does an off-path firewall: the organization's hosts map their EIDs to the EID or IP address of the on-path firewall. Since this setup is functionally the same as today's on-path firewalls that are not explicitly invoked at the IP layer, one might wonder what DOA accomplishes here. The answer is uniformity: in this setup, the configuration of end-hosts is independent of the firewall's placement. Thus, administrators can later move the firewall off-path without reconfiguring every host in the organization.

7 Implementation

We describe our implementation of end-host DOA software, a NEB prototype, and an RPF prototype.

7.1 End-Host DOA Software

In a production deployment, DOA software would be part of kernel protocol stacks, as in §4.4. However, we

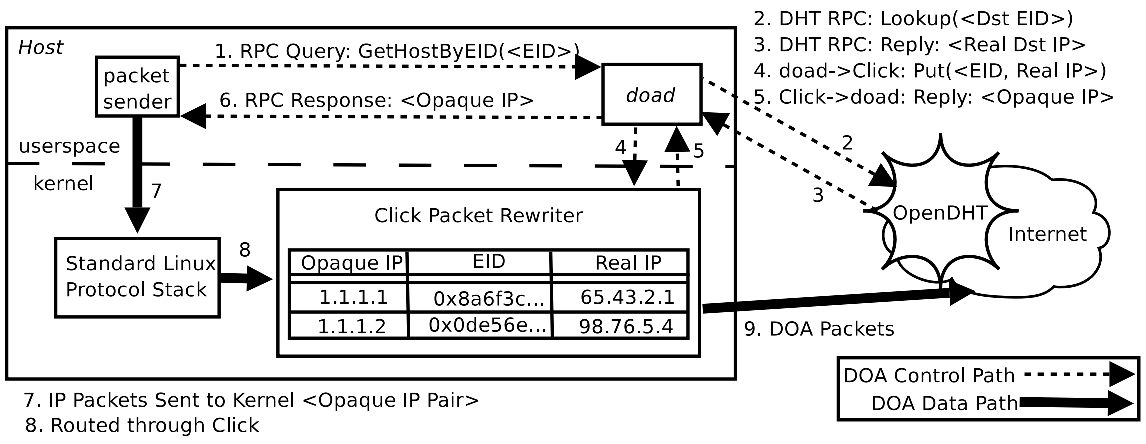
Host
packet sender
1. RPC Query: GetHostByEID(<EID>)
6. RPC Response: <Opaque IP>
doad
2. DHT RPC: Lookup(<Dst EID>)
3. DHT RPC: Reply: <Real Dst IP>
4. doad->Click: Put(<EID, Real IP>)
5. Click->doad: Reply: <Opaque IP>
userspace
kernel
7
4
5
2
3
OpenDHT
Internet
Standard Linux Protocol Stack
8
Click Packet Rewriter
Opaque IP
EID
Real IP
1.1.1.1
0x8a6f3c...
65.43.2.1
1.1.1.2
0x0de56e...
98.76.5.4
9. DOA Packets
DOA Control Path
DOA Data Path
7. IP Packets Sent to Kernel <Opaque IP Pair>
8. Routed through Click

Figure 7: The control and data paths in our prototype implementation of DOA. This figure depicts sending a packet to a given EID obtained out-of-band. Events are numbered in chronological order.

wanted to understand DOA's properties before committing to full kernel implementations, and so we prototyped using a combination of user-level software and Click [31] modules inside the Linux 2.4.20 kernel. Beyond a patch required by Click, we did not modify the kernel. Figure 7 depicts our implementation of end-host DOA software.

Applications get EIDs via user input or DNS and resolve them by invoking the `GetHostByEID` RPC, exposed by *doad*, which is a user-level daemon written in C++ using the SFS libasync library [35]. *doad* implements the RPC by first querying OpenDHT [29] (the key is the EID, *e*; the returned value is the IP address, *i*, that *e* resolves to) and then, via Click's `/proc` file system interface, telling the Click "rewriter" module about the mapping $\langle e, i \rangle$. *doad* returns an opaque handle in the 1.0.0.0/8 subnet that the application uses when the sockets API expects an IP address; the opaque handles allow us to reuse much of the kernel's IPv4, TCP, and UDP software. (Applications could use EIDs instead of the opaque handles if the kernel's networking software were extended to use the `sockaddr_eid` structure, as described in §4.4.) The rewriter module receives IP packets with addresses in the 1.0.0.0/8 subnet, maps these opaque handles to EIDs and real IP addresses and then transmits bona fide DOA traffic. We did not implement EID-to-EID$^+$ mappings.

7.2 NEB Prototype

We implemented (1) a NEB prototype in a Click module and (2) DOA-RIP (§5.5.3) in user space. The NEB has an EID-to-IP table which it uses to rewrite destination IP addresses and which gets an entry when a host behind the NEB, possibly separated by several other NEBs, runs DOA-RIP. After DOA-RIP completes and the NEBs, which also run DOA-RIP, have correct state, the host uses *doad*'s interface to OpenDHT to insert a mapping from the host's EID to the outermost NEB's external IP address, thereby making the host globally reachable.

7.3 RPF Prototype

The RPF is (1) a Click module that associates EIDs to a set of simple rules that are together applied (with OR or AND) to IP-, DOA-, and transport-layer fields to make a "pass" or "fail" decision for each packet and (2) user-level software that communicates with end-hosts, first, to establish a secret key for each EID (using an encrypted, MACed control channel given by the SFS [36] toolkit) and, second, to process requests to add, change, or remove rules. End-host RPF users run (1) a MAC-checking Click module that injects into the kernel's networking stack only those packets that have been correctly MACed by the RPF and (2) user-level software to communicate with the RPF, as just described. The RPF uses HMAC [32] and, in our prototype, it is taken over packet headers, only.

8 Evaluation

The architectural coherence afforded by DOA comes at a performance cost. This section characterizes that cost with microbenchmarks that measure the latency, throughput, and processing time overhead of DOA-enabled data transfers.

8.1 Round-trip Times and Hops

Compared to the status quo, DOA adds network round-trip times. DOA requires an extra resolution—of the EID—when a host first sends a packet to another host. For applications whose end-to-end latency is dominated by DNS lookups (such as Web browsing), the effect of these resolutions might be particularly pronounced. In the most basic DOA configuration—two end-hosts communicating with no off-path intermediaries—the connecting client makes two synchronous network calls: (1) a DNS request to map a human-readable hostname to an EID and (2) a DHT lookup to map a server's EID to its IP address. A third synchronous DHT lookup is required for the server to resolve the client's EID to an IP address.

A recent study [26] indicates that median DNS lookups from a network at MIT can vary from about 70 ms in the case of NS server cache hits, to about 190 ms in the case of cache misses. By contrast, we measured median DHT lookups of random EIDs stored in OpenDHT at 138 ms. Thus, DOA can add noticeable delays to small data transfers, sometimes tripling their end-to-end latencies.

However, for latency-sensitive hosts and applications, the following optimizations are possible:

- The DHT could use Beehive's [45] proactive, model-driven caching strategy to reduce the number of network round-trips required by lookups to an average of one or less than one (assuming the request pattern for EIDs is heavy-tailed).
- DNS names of hosts could resolve to the EID *and* the `erecord` (or to the chain of `erecord`s that together indicate how to reach the host), thereby requiring one DNS lookup, as under the status quo, to send a packet to a host. In this case, DNS itself would be caching `erecord`s.
- To save a remote host the burden of an EID resolution when responding to an initiating host, the initiating host could send its `erecord` (as noted in §4.2).

In addition, DOA adds network hops: when a packet travels from a source to an off-path middlebox en route to a destination, the packet (in most cases) takes more network hops than if it had traveled from source to destination directly. This extra latency is inevitable if one wants to invoke an off-path intermediary. There is no "correct" trade-off between latency and the flexibility of off-path functions; different users have different preferences.

8.2 Packet Size Overhead

DOA packets in our implementation have a 68-byte DOA header (the 44 bytes shown in Figure 2 plus 24 for the DOA security header mentioned in §6.1). This overhead affects the maximum number of packets per second sustainable by DOA senders and receivers and is more costly for smaller packet payloads. For example, adding a DOA header to a 1466-byte UDP-over-IP-over-Ethernet packet (the UDP payload here is 1400 bytes) increases the packet size by 4.6%. For 130-byte packets (with UDP payload of 64 bytes), the 68 bytes of DOA header add overhead of more than 50%.

For large packets, this overhead is likely to bottleneck DOA's sustainable throughput. To verify this claim, we now characterize the throughput our DOA implementation can sustain when sending and receiving large packets. We measure both DOA and non-DOA traffic and find that the packet header overhead introduced by DOA explains the throughput difference between the two cases. Each measurement below is the average of five trials involving 1 GB of data, and the average packet drop rate was less than 1%. Our experiments do not involve the DHT; prior to the experiments, we resolved the destination EIDs to IP addresses.

Component	cycles/pkt	μs/pkt
DOA$\rightarrow$IP	1894.2	1.11
filter	9410.3	5.54
verify	8773.8	5.16

Table 1: Processing time per packet for DOA components. The first column contains the number of cycles, while the second column contains the calculated time (in μs) needed to perform that number of cycles on an Intel Celeron 1.7 GHz processor.

To get a baseline, we measured the number of UDP packets per second that one of our test hosts can send another. We tested large packets (1400-byte payloads) over a Gigabit Ethernet network, tuned the sending rate to achieve maximum throughput, and measured the number of packets that exited the receiver's device driver queue. On average, the receiver processed 72900 packets per second, or 778.5 Mbit/s. The bottleneck here appears to be our hosts' PCI buses.

Next, we ran the same test with DOA packets. The sender uses our end-host DOA software (§7.1), which inserts a DOA header into IP packets and rewrites IP headers. The receiver performs a similar process to translate DOA packets to IP packets. In this case, the receiver processed 69600 packets per second, or 743.0 Mbit/s, which is 4.6% slower than the baseline. We conclude that the slowdown here is due entirely to packet size overhead. These tests were for large packets; as noted above, small packets will be much more penalized by this overhead.

8.3 Processing Time

For small packets, however, in addition to packet size overhead, CPU costs for per-packet DOA operations will limit the rate at which DOA hosts can process packets. We now characterize this potential bottleneck on small packets (64-byte UDP payloads). Using Click tools to read the processor's cycle counter before and after our DOA modules, we estimated the number of cycles used by the modules; the reported numbers are averages over 80000 processed packets. Note that these numbers are upper bounds on average processing time: the implementation is untuned, and our measurements include cycles consumed by interrupt handling for other kernel processes. Table 1 summarizes our observations.

We first measured the processing time needed by our receiver's DOA software, which translates DOA packets to IP packets (labeled "DOA→IP" in Table 1). This component takes nearly 1900 cycles—or 1.11 μs on the host we used for testing, which has an Intel Celeron 1.7 GHz CPU—to process each packet.

We next considered the processing time for the operations associated with an RPF (§6), namely HMAC [32] computation and verification. In our experiments, the RPF operates as described in §7.3; it uses a single default rule that passes all packets intended for the receiver and holds a mapping from the receiver's EID to an IP address. The RPF computes the MAC for each packet, writes the MAC to the DOA header, and forwards the packet to the receiver. When the receiver gets the packet, it verifies the MAC before passing the packet to its end-host DOA software. As shown in Table 1, the filter takes more than 9400 cycles (5.54 μs) to apply its rule and compute a MAC, and the receiver takes nearly 8800 cycles (5.16 μs) to verify the MAC.

8.4 Discussion

Of the three types of costs imposed by DOA—lookup latency, packet size, CPU overhead—the latter two would only appear to users under the most stressful system conditions. The first cost, latency, is serious because, absent optimizations, it is visible to end-users. However, as noted in §8.1, there are several caching strategies that substantially mitigate, if not eliminate, this latency. Ultimately, we believe that the costs imposed by DOA are outweighed by the benefits of a coherent framework for reasoning about, and deploying, intermediaries.

9 Related Work

Besides the direct influence of i3 [57], HIP [39–41], and UIP [17] on our mechanisms and insights, an older proposal for location-independent EIDs [34] grew out of Nimrod [9]. Shoch [52] and Saltzer [50] have been among many (see [10, 11, 13, 19, 33, 42, 43] and references therein) to distinguish between network elements' identity and location. Indeed, much of what we mention below separates these two concepts, usually by creating a set of end-host identifiers distinct from network location.

i3 canonizes this separation with an infrastructure that uses flat identifiers in packets to decouple sending (into the infrastructure) and receiving (from the infrastructure). These identifiers name services whereas EIDs name hosts. Like DOA, i3 is specifically designed for senders and receivers to invoke intermediaries. i3 does not hold the proliferation of private addressing realms as a principal concern, but one can leverage i3 to reach machines behind NATs without modifying or configuring the NATs [28]. The main difference is that the DOA architecture requires a *resolution* infrastructure while i3 depends on a *forwarding* infrastructure; under the pure design, all i3 packets are sent into the infrastructure.

TRIAD [22] is an extension to the Internet that addresses many architectural ills, including NAT. TRIAD hosts receive location-independent names. As in DOA, these names may resolve to a logical path, and IPv4 addresses are routing tags without end-to-end significance. TRIAD does not focus on a framework for network-layer middleboxes, though its mechanisms can certainly accommodate them, and the authors give a solution for NAT traversal. The technical details of our approaches differ: TRIAD names are derived from domain names (in contrast to flat EIDs), and under TRIAD, resolution and routing are conflated, thereby improving latency.

HIP also separates location and identity; its goal is architectural support for mobility and multi-homing. DOA borrows some of HIP's mechanisms and applies them to middlebox issues, which is not HIP's focus.

In contrast, some work is expressly motivated by the proliferation of private addressing realms. UIP [17], from which we also borrow, creates an overlay among participating hosts to interconnect heterogeneous or NATed networks. Like DOA, UIP incorporates HIP-style flat host identifiers. UIP hosts form an ad-hoc overlay by using a DHT-inspired algorithm to route packets for each other based on the destination identifier. Our approach contrasts with UIP's in that, while both projects address middleboxes' proliferation, we focus on an architecture that explicitly welcomes middleboxes whereas UIP's overlay of peers makes them transparent. IPNL [20] is an extension to the Internet architecture that solves problems resulting from private addressing realms. IPNL relies in part on bona fide host identifiers; these identifiers are domain names, though the authors acknowledge the security benefits of HIP-style flat identifiers. Like DOA, IPNL tries to coherently incorporate NATs into the Internet architecture, and both designs modify hosts and NATs but not IPv4 routers.

Other projects have tried to obsolete middleboxes; these run the gamut from architectural enhancements to radical reorganizations. An example in the middle of this spectrum is IPv6 [15]. IPv6 addresses are globally unique (thus addressing one motivation for NATs), but, as noted in §3.2, do not satisfy the requirement of topology-independence. Predicate Routing [47] and network capabilities [1] propose architectural enhancements for security and denial-of-service protection. Radical network architectures and meta-architectures include Role-Based Network Architecture [7] and FARA [10]. Our approach contrasts with these because, first, our goal is explicit invitation of middleboxes and, second, these proposals, if fully realized, require at least some changes to all network elements, not just hosts and middleboxes.

In contrast, other work has avoided creating identifiers for end-hosts but has nonetheless accepted middleboxes as an architectural problem to be worked around. MIDCOM [56, 58] is a protocol and framework intended to remove intelligence from NATs and firewalls by offloading application-specific behavior to designated agents, which insert dynamic state into intermediaries automatically. For example, in response to a globally reachable host initiating an Internet telephony session to a NATed host, the agent would ask the NAT to open the appropriate destination UDP port and would close the port at session's end. Like DOA, MIDCOM aims to simplify management of NATs and firewalls by creating state automatically. However, because MIDCOM focuses only on application- and not networking- and transport-level software, persistent host identifiers are unavailable to them, and thus their protocols devote considerable energy (and complexity) to handling the overloading of port fields. Also, MIDCOM's techniques work through only one layer of NAT [56] in contrast to our supposition that hosts may be behind several layers.

Twice NAT [55], Realm Specific IP [6, 55], and STUN [48] all address specific problems posed by NATs. A recent Internet draft [18] summarizes various techniques by which P2P applications can handle middleboxes. While useful for the current network architecture, these (largely manual) tactics for exposing NATed hosts would be unnecessary if all hosts had location-independent identifiers. Today, many home users attempt to create persistent identifiers for frequently renumbered hosts with Dynamic DNS, *e.g.*, [16]. Since DNS names are resolved to IP addresses and are not carried in packets, they are quite useful as naming handles for humans but not for network elements.

DOA's use of the delegation primitive to simplify firewalls is preceded by a body of literature that addresses the error-prone and time-consuming nature of firewall configuration. The Firmato toolkit [4], for example, takes a language-based approach to simplifying firewall configuration by abstracting away low-level configuration details. Distributed firewalls [5,25,30] take simplification one step further: a centralized, managed entity downloads firewall rules to end-hosts (which it identifies with IPSEC certificates in analogy with our use of EIDs to associate policies with hosts). In contrast to the approach described in §6, distributed firewalls do not off-load from clients the job of actually applying rules.

10 Conclusion

The Internet architecture was defined in a context where traffic was benign and addresses plentiful. There were no reasons to interpose functions other than forwarding between endpoints, which became the end-to-end rallying cry for the architecture. Today's Internet is a very different place. There are many reasons why users interpose functions that, in the canonical architecture, either belonged on their host (such as firewalls) or didn't belong at all (such as NATs). The Internet architecture was not designed for—in fact, one might say it was designed against—such interposition of function.

The current incarnations of interposition, middleboxes, are widely derided for their violations of the architecture and the resultant loss of flexibility in the Internet. However, the complexity and risk associated with being a network host, which used to be minimal, is now daunting even to expert users. We therefore expect outsourcing functionality to become increasingly common.

The architecture presented in this paper, DOA, has a simple goal: to allow the Internet to reap the benefits of network-level middleboxes without their harmful side-effects. It does so not by altering IP, or routers, but by making delegation a basic primitive and introducing a set of globally unique endpoint identifiers.

Acknowledgments

We are grateful to Russ Cox, Bryan Ford, Frans Kaashoek, Karthik Lakshminarayanan, David Mazières, the anonymous reviewers, and our shepherd, Geoff Voelker, for their excellent comments, which substantially improved this paper. Sean Rhea and John Bicket gave useful help with OpenDHT and Click, respectively. We thank Karthik Lakshminarayanan, Sylvia Ratnasamy, and Ion Stoica for useful conversations about naming in the Internet architecture. This work was supported by the NSF under Cooperative Agreement No. ANI-0225660, a Sloan Foundation fellowship, an MIT EECS fellowship, an NSF graduate fellowship, and an NDSEG fellowhip.

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *2nd ACM Workshop on Hot Topics in Networks*, Cambridge, MA, Nov. 2003.

[2] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, Feb. 2003.

[3] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the Internet. In *ACM SIGCOMM*, Portland, OR, Aug. 2004.

[4] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. *Firmato*: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, May 1999.

[5] S. M. Bellovin. Distributed firewalls. *;login: Magazine, Special Issue on Security*, Nov. 1999.

[6] M. Borella, D. Grabelsky, J. Lo, and K. Taniguchi. Realm Specific IP: Protocol specification, Oct. 2001. RFC 3103.

[7] R. Braden, T. Faber, and M. Handley. From protocol stack to protocol heap – role-based architecture. In *1st ACM Workshop on Hot Topics in Networks*, Princeton, NJ, Oct. 2002.

[8] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues, Feb 2002. RFC 3234.

[9] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod routing architecture, Aug 1996. RFC 1992.

[10] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the addressing architecture. In *SIGCOMM FDNA Workshop*, Karlsruhe, Germany, Aug. 2003.

[11] D. Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing reality: An architectural response to demands on the evolving Internet. In *ACM SIGCOMM FDNA Workshop*, Karlsruhe, Germany, Aug. 2003.

[12] D. D. Clark. The design philosophy of the DARPA Internet protocols. In *ACM SIGCOMM*, Stanford, CA, Aug. 1988.

[13] M. Crawford, A. Mankin, T. Narten, I. J. W. Stewart, and L. Zhang. Separating identifiers and locators in addresses: An analysis of the GSE proposal for IPv6, Oct. 1999. Internet draft

`draft-ietf-ipngwg-esd-analysis-05.txt` (Work in progress).

[14] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM SOSP*, Banff, Canada, Oct. 2001.

[15] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6), Dec. 1998. RFC 2460.

[16] Dynamic Network Services, Inc., http://www.dyndns.org.

[17] B. Ford. Unmanaged Internet Protocol: taming the edge network management crisis. In *2nd ACM Workshop on Hot Topics in Networks*, Cambridge, MA, Nov. 2003.

[18] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer (P2P) communication across middleboxes, Oct. 2003. Internet draft `draft-ford-midcom-p2p-01.txt` (Work in progress).

[19] P. Francis. *Addressing in Internetwork Protocols*. PhD thesis, University College London, UK, 1994.

[20] P. Francis and R. Gummadi. IPNL: A NAT-extended Internet architecture. In *ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[21] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A framework for IP based virtual private networks, Feb. 2000. RFC 2764.

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

[23] T. Hain. Architectural implications of NAT, Nov. 2000. RFC 2993.

[24] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol, Mar. 1999. RFC 2543.

[25] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *Computer and Communications Security*, Nov. 2000.

[26] J. Jung, Feb. 2004. Personal communication. Data was gathered at MIT using the method described in [27].

[27] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS peformance and the effectiveness of caching. *IEEE/ACM Trans. on Networking*, 10(5), Oct. 2002.

[28] J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Supporting legacy applications over i3. Technical Report UCB/CSD-04-1342, UC Berkeley, May 2004.

[29] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring adoption of DHTs with OpenHash, a public DHT service. In *3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2004.

[30] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. The STRONGMAN architecture. In *Third DARPA Information Survivability Conference and Exposition*, Apr. 2003.

[31] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, Aug. 2000.

[32] H. Krawzyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication, Feb. 1997. RFC 2104.

[33] E. Lear and R. Droms. What's in a name: Thoughts from the NSRG, Sept. 2003. draft-irtf-nsrg-report-10, IETF draft (Work in Progress).

[34] C. Lynn. Endpoint Identifier Destination Option. Internet Draft, IETF, Nov. 1995. (expired).

[35] D. Mazières. A toolkit for user-level file systems. In *Proc. 2001 Usenix Technical Conference*, June 2001.

[36] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proc. 17th ACM SOSP*, Kiawah Island, SC, Dec. 1999.

[37] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley; 2nd edition, 1996.

[38] K. Moore. Things that NATs break, http://www.cs.utk.edu/˜moore/opinions/what-nats-break.html, as of Oct 2004.

[39] R. Moskowitz and P. Nikander. Host identity protocol architecture, Sep 2003. draft-moskowitz-hip-arch-05, IETF draft (Work in Progress).

[40] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host identity protocol, Oct 2003. draft-moskowitz-hip-08, IETF draft (Work in Progress).

[41] P. Nikander, J. Ylitalo, and J. Wall. Integrating security, mobility, and multi-homing in a HIP way. In *Network and Distributed Systems Security Symp. (NDSS '03)*, San Diego, CA, Feb 2003.

[42] M. O'Dell. 8+8 - An alternate addressing architecture for IPv6, Oct. 1996. Internet draft `draft-odell-8+8-00` (Work in progress).

[43] M. Ohta. 8+8 addressing for IPv6 end to end multihoming, Jan. 2004. Internet draft `draft-ohta-multi6-8plus8-00` (Work in progress).

[44] R. Perlman. Understanding ikev2: Tutorial, and rationale for decisions, Feb. 2003. Internet draft `draft-ietf-ipsec-ikev2-tutorial-01.txt` (Work in progress).

[45] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, Mar. 2004.

[46] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private Internets, Feb. 1996. RFC 1918.

[47] T. Roscoe, S. Hand, R. Isaacs, R. Mortier, and P. Jardetzky. Predicate routing: Enabling controlled networking. In *1st ACM Workshop on Hot Topics in Networks*, Princeton, NJ, Oct. 2002.

[48] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN – simple traversal of user datagram protocol (UDP) through network address translators (NATs), Mar. 2003. RFC 3489.

[49] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, Heidelberg, Germany, Nov. 2001.

[50] J. Saltzer. On the naming and binding of network destinations. In P. Ravasio et al., editor, *Local Computer Networks*, pages 311–317. North-Holland Publishing Company, Amsterdam, 1982. Reprinted as RFC 1498, August 1993.

[51] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), Nov. 1984.

[52] J. F. Shoch. Inter-network naming, addressing, and routing. In *17th IEEE Computer Society Conference (COMPCON '78)*, Washington, DC, Sept. 1978.

[53] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. ACM MOBICOM*, 2000.

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

[55] P. Srisuresh and M. Holdrege. IP network address translator (NAT) terminology and considerations, Aug. 1999. RFC 2663.

[56] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox communication architecture and framework, Aug. 2002. RFC 3303.

[57] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[58] R. P. Swale, P. A. Mart, P. Sijben, S. Brim, and M. Shore. Middlebox communications (MIDCOM) protocol requirements, Aug. 2002. RFC 3304.

[59] Symantec Corporation. Norton Personal Firewall, http://www.symantec.com/sabu/nis/npf/, as of Oct 2004.

[60] VMWare, Inc. `http://www.vmware.com`.

[61] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, Mar. 2004.

[62] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.