Prototyping a DHT-Oriented Architecture

Maxwell Krohn, Jeremy Stribling, Michael Walfish
{krohn, strib, mwalfish}@csail.mit.edu

Abstract

We attempt to decouple identity from location in Internet hosts. In our proposal, hosts receive flat identifiers in a large and sparse namespace, and an Internet-wide distributed hash table (DHT) acts as a resolver by mapping these flat identifiers to IP addresses in analogy with today's DNS. Unlike DNS names, the source and destination identifiers appear in packets, in a shim layer after the IP header. Our proposal would change all host software but leave the core routers untouched. The advantages of our proposal are: (1) the benefits of decoupling location from identity, which have been articulated elsewhere (see, for example, [16, 19]) and (2) enhanced middlebox functions. The paper's focus is the second of these two advantages.

1 Introduction

Creating a set of *host identifiers* in the Internet is an old goal, and we share it. Proposals for host identifiers have occupied a wide spectrum [16]; for our purposes, host identifiers are strings that appear in ordinary packets to indicate from which entity a given packet originated and to which entity it is destined. IP addresses—the only set of global identifiers carried in packets on the Internet—actually describe network *locations*. No set of identifiers on the Internet exists that can robustly identify individual *hosts*. The motivation for host identifiers, separate from IP addresses, is twofold:

IP renumbering: First, as has been well articulated in the literature (see [16, 19] and references therein for a review of this topic), host identifiers could solve the IP renumbering problem, which results from the simple fact that a given host's IP address can change. Specific reasons for renumbering include: (1) the host is moving (referred to as "mobility") (2) the host has several logical interfaces, each connected to different network providers ("multihoming") or (3) the host receives its IP address through DHCP ("dynamic addressing"). The problematic implications of renumbering are, first, transport connection breakage when the IP address of either communicating party changes and, second, semantic limitations resulting from the fact that an IP address cannot be stored and later used as a persistent reference to a given host.

Accommodating middleboxes: Second, as observed in a recent proposal [1], whose motivation we follow closely, a set of host identifiers would be an architectural primitive for solving problems related to *middleboxes*. We define a *middlebox* as a network element, other than the ultimate source or destination of an IP packet, that performs a function on the packet other than pure IP routing. Examples of middleboxes are firewalls, VPNs, and NATs [26].[1] Although [1] considers a wide class of middleboxes, in our work here we consider only NATs. A NAT allows many hosts on the same private subnet, behind the NAT, to share a single external IP address. The problematic implication of this multiplexing is that hosts outside a private addressing realm may be unable to initiate connections to hosts within the private addressing realm, even if all concerned parties wish to permit this type of communication.

We believe that a set of host identifiers, carried in packets, would address the problems mentioned above.[2] For reasons articulated in [1], we believe that an architectural proposal like this one should avoid router modification. We thus limit ourselves to changing packets only after IP headers and modifying only host and middlebox software. We also aim to retain current logic in application software and therefore seek to retain current application interfaces to transport protocols. The logical solution to both of these constraints (not modifying IP and preserving application interfaces to current trans-

[1] Throughout this paper, we used the term "NAT" to mean either NAPT or NAT.

[2] We do not claim that all share our opinion about whether the problems mentioned are worth solving or are indeed problems; we claim only that to the extent one believes they are problems, a set of host identifiers would in fact address them.

port protocols) is placing host identifiers in packets between the IP header and transport header. TCP connections logically bind to the host identifiers, and not to the underlying IP address; the underlying IP address is reduced to a routing descriptor.

For reasons again mentioned elsewhere [1], we believe that the host identifiers should be flat, unreadable strings in a large and sparse namespace. The existing IP infrastructure should not *route* on these host identifiers; our intent is rather to have sending (receiving) entities *resolve* these host identifiers to IP addresses when actually sending (receiving) packets. This *resolution* suggests distributed hash table (DHT) technology, a decentralized solution precisely designed for scalable resolution in a large and sparse namespace.

The purpose of this project is to prototype an architecture that incorporates the elements mentioned above. Although we believe our prototype would address the IP renumbering problem, for the remainder of the paper we focus only on accommodating NATs. This paper describes our prototype of the following:

- a shim layer in packets after the IP header and before the transport layer to hold source and destination host identifiers;
- host software to interact with an Internet-wide DHT that holds identifier-to-IP address mappings;
- a simple NAT that uses the host identity layer to permit new functions, like allowing entities to initiate connections on arbitrary TCP ports to machines behind the NAT;
- a protocol, requiring no manual steps, that "punches holes" in an arbitrary tree of NATs so that hosts in the global Internet can reach hosts behind multiple layers of NATs.

2 Scenario and Assumptions

We assume throughout this paper the following:

- that the Internet is organized as a tree of private address spaces, rooted at *the core*;
- that the core is reachable from all Internet hosts, including those within private address spaces;
- that a widely deployed distributed hash table (DHT) infrastructure exists, is in the core, and is thus globally reachable.

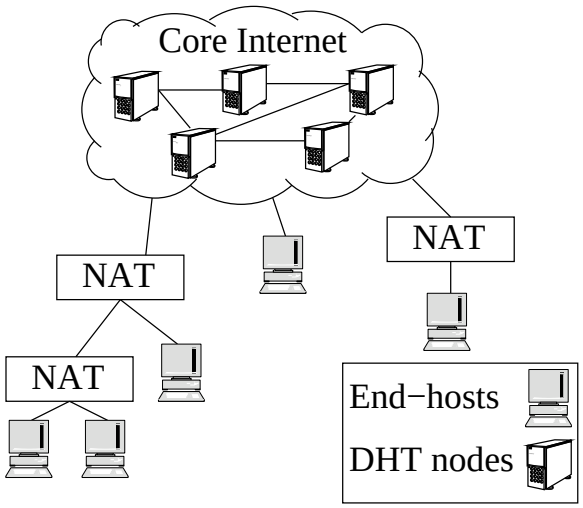

Figure 1: Our assumed network scenario: a global Internet core reachable by all hosts and a DHT deployed within this core. Hosts can be directly connected to the core or can be connected through an arbitrary number of NATs.

This scenario is depicted in Figure 1. An example of several layers of NATs is the case in which a virtual host runs behind a virtual NAT, and these virtual machines together run on an actual host which is physically attached to a NAT in someone's private home network. Continuing the example, the physical NAT is connected via a cable modem to the core, and so the physical NAT's uplink port has a globally reachable IP address.

3 Design of DOA

We follow [1] and call it—and this project—DOA (because both [1] and we advocate a DHT-Oriented Architecture). The authors of [1] refer to host identifiers as *EIDs* (End-Point Identifiers). These EIDs are 160 bits and, as mentioned in the introduction, they are carried in IP packets, just after the IP header but before the transport header. We assume a one-to-one mapping between hosts and EIDs and that users will get out-of-band (*e.g.*, via a directory service or DNS) the EIDs of hosts they are trying to contact. Figure 2 depicts the major components we envision and the interfaces between them. The remainder of this section discusses these components.

3.1 DOA Packets

The protocol layer we are adding in between transport and IP is called DOA. DOA packets are delivered over IP, with the IP protocol field set to 203 (an

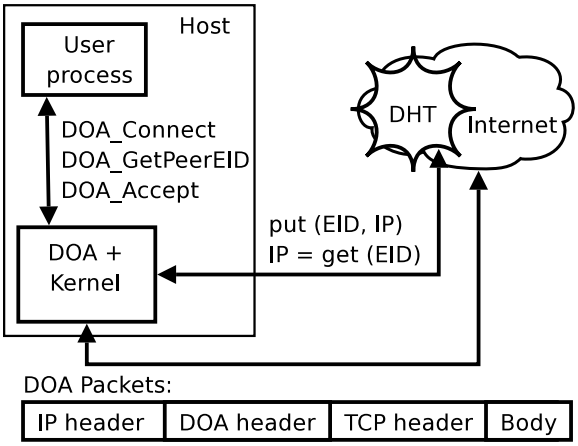

Figure 2: Overview of DOA architecture.

unassigned IP protocol as of May 2004). The DOA header is depicted in Figure 3, and the fields therein are as follows: the version is 0 but can change; the header length measures the length of the DOA frame header in words; the "protocol" field specifies what transport-level protocol is encapsulated by the given packet; the last 16 bits give the total length of the DOA packet, in bytes. The DOA packet will usually have a header size of 11 words. One word is used for the header, five words for the source EID, and five more words for the destination EID.

3.2 Resolution Substrate

End-hosts resolve EIDs to IP addresses by querying a *resolution substrate*. The resolution substrate could be any global service offering a simple put()/get() interface. In our design, the lookup service is a globally-reachable DHT infrastructure that provides robust and scalable resolution. To protect a given host's EID against unauthorized modification, we use self-certification: EIDs must be the hash of a public key (and optionally a salt). Entities who do get() operations on the DHT should ensure that the returned IP address is signed with the private key whose corresponding public key was hashed to create the EID.

3.3 Host Software

This subsection describes the logical API to DOA seen by applications running on DOA-enabled hosts. The interface that DOA actually exports in our prototype differs from this logical API and is discussed in Section 5.

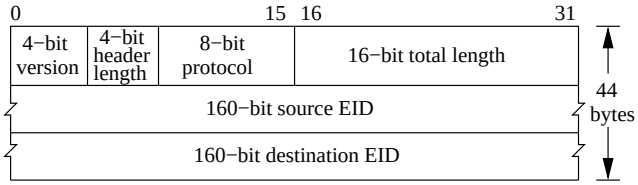

Figure 3: The DOA Packet Header.

3.3.1 Logical API

- `fd = DOA_Connect(EID, port)`: The application supplies this function with the EID (obtained out-of-band) of the desired connection endpoint. The application also supplies the (TCP or UDP) port to which it would like to connect. In response to this call, the DOA software issues a DHT lookup to resolve the EID into an IP address.[3] If the resolution fails, this function returns an error.
- `fd = DOA_Accept(sockfd, &EID, &port)`: A server accepts connections on the given socket `sockfd` using this function. On successful return, `EID` and `port` are set to the identifier and port of the connecting host.
- `EID = DOA_GetPeerEID(fd)`: This function returns the EID of a remote, connected entity.

3.3.2 Sample API Usage

We believe that using this API to construct DOA-enabled applications will be straightforward for developers. In place of the usual `connect` system calls, they will instead call `DOA_Connect` using an EID they have obtained out-of-band (perhaps by having resolved a DNS name). On the other end of the connection, a server will have been listening on a certain port and will be accepting connections via `DOA_Accept`. When the `DOA_Accept` call returns, the server will know the EID of the connecting host; it may also retrieve the EID later using the `DOA_GetPeerEID` call, in analogy with the current `getpeername` system call.

4 Design of NATs under DOA

We now present an approach to making end-hosts behind layers of NATs reachable under DOA.

[3] The host software might have cached the EID-to-IP mapping and thus would not do a DHT lookup.

4.1 DOA NATs Need Per-Host State, Only

NATs, under DOA and under the status quo, must *demultiplex, i.e.*, they must replace the destination IP address of packets sent to their own IP address with the private IP address of the packet's ultimate destination. Whereas today's NATs demultiplex by using the IP address and port fields plus per-connection state that the NAT creates opportunistically (in response to connections initiated by hosts behind the NAT, for example), a NAT box under DOA need neither maintain per-connection state nor overload existing protocol fields.

Under DOA, the NAT gets the private IP address—which replaces the original destination IP address in the packet—by looking up the packet's destination EID in a static EID-to-IP lookup table maintained by the NAT. A mapping from an EID to an IP is also called a *route.* For packets that originate behind the NAT, the NAT replaces the source IP address with its own.

4.2 Automatic and Secure NAT Hole Punching

As we discussed in Section 2, we imagine that end-hosts are leaves in a tree whose internal nodes are NATs, and as we discussed in the Introduction, a principal goal of this project is exposing hosts behind NATs to other hosts. However, consider the following situation: a host, h, outside the private address space of an end-host, e, wants to gain access to e. In order for this communication to succeed, all of the NATs on the path between h and e must have static entries in their lookup table that map the EID of e to the IP address of either the end-host or the next NAT in the path of NATs. Creating this static state in the NATs is a challenge: we would like a protocol that is at once automatic and secure. Below we define what "secure" means.

4.2.1 Security Assumptions and Threat Model

End-hosts and internal nodes (*i.e.*, NATs) inside a tree of NATs must naturally trust all NATs upstream of them; our design provides machines with no recourse if an upstream NAT is compromised or dishonest. We assume that an end-host can acquire, via a trusted out-of-band channel, the EID of its *gateway*, defined as its parent in the tree. End-hosts trust neither other end-hosts nor downstream NATs. We also assume all links in the NAT tree are vulnerable to eavesdropping and tampering.

4.2.2 Security Goals

Recall that we are searching for a protocol that allows an end-host to establish state at each NAT on the path from itself to the globally-reachable core. We require the protocol to satisfy these security properties:

1. A mapping from EID to private_ip must be authorized by the party who holds the private key corresponding to EID.
2. If all NATs along the path from the end-host to the core behave honestly, then the path established is correct.

Property 1 prevents unauthorized hosts from spoofing routes and diverting traffic from its rightful destination. Property 1 also implies resilience to a form of replay attack. IP address reassignment is common in private networks. If host A is assigned host B's old IP address, A should not be able to replay B's protocol messages to steal B's traffic. Rather, protocol messages must expire after they are processed so that hosts cannot rollback routes without authorization.

Property 2 is a response to potential man-in-the-middle attacks during route establishment. Consider a host A that learns about host B's attempt to establish a path to the core. Even if A can tamper with packets between B and the core, it should not be able to insert itself into B's route. That is, we require the route establishment protocol to be "correct," meaning the logical path it establishes to the core should be topologically equivalent to the direct physical path through the NAT network.

We emphasize that our goal is securing the control and not the data plane of DOA NATs; an attacker can always tamper with data traffic.

4.2.3 DOA-RIP

We present DOA-RIP (DOA Route Implementation Protocol) and argue it meets the two security goals above. DOA-RIP is a two-round protocol initiated by a NATed end-host that wants to make itself reachable. The first round of the protocol creates a traceroute, signed by all NATs along the host's path to the core. The second round propagates the state to these NATs so they can make appropriate insertions in their EID-to-IP lookup table.

Assume that an end-host has EID e_0 and IP address i_0. Assume that there are n NATs between

e_0 and the core. The NAT closest to the end-host has EID e_1 and IP address i_1. Similarly, the NAT connected to the core has EID e_n and IP address i_n. Before the protocol begins, we assume that a host knows the public key of its parent NAT.[4] The protocol is as follows.

Round 1: Secure Traceroute

1. e_0 sends an initialization message with its EID to e_1; for all $1 \leq k < n$, the NAT e_k forwards the initialization message to NAT e_{k+1}.

2. NAT e_n receives the initialization message. It then picks random nonce r_n, and associates it with the EID e_0 in a local cache. The NAT then constructs a message $x_n = \langle e_n, i_n, r_n \rangle$. It signs x_n with its private key and sends x_n and its signature to e_{n-1}.

3. For all $1 \leq k < n$, the NAT e_k receives the message x_{k+1} from NAT e_{k+1} and verifies the signature using e_{k+1}'s public key. If this verification succeeds, NAT e_k likewise picks a random nonce r_k and associates it with e_0 in its local cache. It appends the triple $\langle e_k, i_k, r_k \rangle$ to the message x_{k+1} and calls this new message x_k. It signs x_k and sends both the message and the signature to e_{k-1}.

4. When the end-host e_0 receives x_1, it verifies the message using e_1's public key. If the verification succeeds, then the end-host has a complete traceroute to the core.

Round 2: Routing rule propagation

1. For $1 \leq k \leq n$, the end-host prepares a route insertion request $y_k = \langle e_0, i_{k-1}, r_k \rangle$. It signs each request individually with its private key. It then concatenates all requests, appends its public key, and sends this package up the NAT chain.

2. For $1 \leq k \leq n$, the NAT e_k receives the route insertion request y_k. It verifies the signature of this message using the public key whose hash is e_0. It verifies that the nonce in y_k is indeed the nonce r_k it previously issued for EID e_0.

   If these two verifications succeed, the NAT will insert the rule $\langle e_0, i_{k-1} \rangle$ into its routing table and will propagate the insertion request up the NAT chain. It then clears the nonce r_k from its cache. If NAT e_k does not hear a valid y_k message within T seconds of issuing its nonce in the first round, it times out r_k and flushes its cache. In our implementation, T is set to 10.

Assuming that all steps succeed, DOA-RIP establishes correct "back pointers" at all NATs between an end-host and the core, as we desired.

4.2.4 Security Discussion

We argue informally that DOA-RIP has security property 1 since the EID is self-certifying. The nonces ensure that the self-certified route insertion request expires immediately after use and therefore cannot be replayed. The timeout mechanism described in Round 2, Step 2 protects against the attack in which host A delays host B's messages and then releases B's messages once it has control of B's local IP address.

DOA-RIP securely creates correct routes, as Property 2 requires, because each host e_k knows the EID of the NAT e_{k+1} directly upstream of it. Each host can therefore verify that the traceroute message received in Step 3 of the first round is from the authorized upstream NAT, not from a malicious man-in-the-middle. The entire path from the host to the core is therefore correct, by induction.

DOA-RIP currently assumes that the topology of the NAT tree is stable and that end-hosts clean up when they leave the network. To be more robust, the NATs should periodically time out lookup entries, and end-hosts should periodically refresh them. A study of DOA-RIP under route and host instability is left to future work.

5 Implementation

5.1 Core DOA Implementation

It goes almost without saying that in a production deployment of DOA, software for processing DOA packets would be integrated into the kernel's networking software, applications would traffic in EIDs, and the logical API (Section 3.3.1) would be the actual API. However, our instantiation of DOA was not intended to be a production deployment. Our implementation goals were instead maximizing compatibility with legacy applications and de-

[4] A host can obtain and verify its parent's public key given the parent's EID, which is known, as assumed in Section 4.2.1.

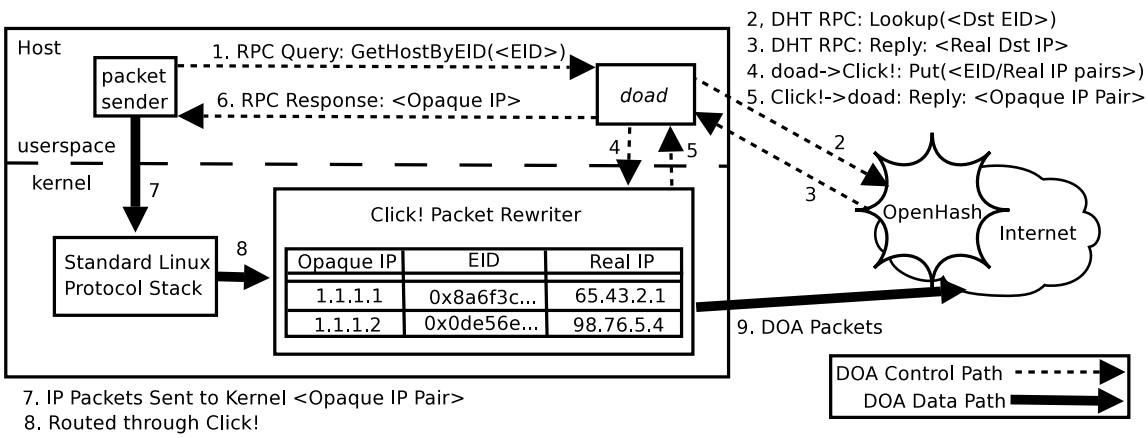

Figure 4: The control and data paths in our prototype implementation of DOA. This figure depicts sending a packet to a given EID obtained out-of-band. Events are numbered in chronological order.

veloper convenience. Our prototype satisfies these goals as follows:

- Making a host DOA-enabled requires only three steps: (1) installing *doad*, a user-level daemon (2) patching the kernel to accommodate Click! [15] and (3) installing a Click! *rewriter module*;
- The kernel's networking software is blissfully unaware of DOA. Functions to create, process, and consume DOA packets are implemented by the Click! rewriter module and executed, in the host-to-network direction, only after the kernel washes its hands of the packet.[5]
- The software we expose to applications mostly hides EIDs; when using the sockets API, which we do not modify, both applications and the kernel traffic in opaque handles that bear a suspicious resemblance to IP addresses in the 1.0.0.0/8 subnet; these opaque handles are later mapped to EIDs by the Click! module.

The control and data planes for our implementation are depicted in Figure 4. The purpose of the control plane is to use a DHT to resolve a given EID, obtained by the application out-of-band, to the globally reachable IP address of the end point with which the application is trying to communicate. The application invokes the control plane via the `GetHostByEID` RPC, which is exposed by *doad*. The return value of this RPC is an opaque handle, OIP, that the application treats as an IP address and that *doad* had obtained from the packet rewriter module.

The data plane, at a high-level, works as follows: in the host-to-network direction, for example, the application hands OIP (which is an IP address of the form $1.x.y.z$) to appropriate calls in the socket API (*e.g.*, `sendto()` or `connect()`). This use of OIP results—owing to an entry in the host's routing table inserted by the Click! configuration—in the data path depicted in Figure 4. Namely, after the kernel forms a complete non-DOA IP packet, the Click! module turns it into a DOA-over-IP packet and sends the packet into the network. More detail follows.

5.1.1 Application Interface to DOA: *doad*

doad listens on a Unix domain socket for `GetHostByEID` RPC calls from local applications. The daemon queries the OpenHash [14] DHT via Sun RPCs to a nearby well-known gateway, as specified by the OpenHash interface. Communication between *doad* and the packet rewriter is always initiated by *doad* and happens via control messages of the form "Create an entry for EID; EID's real IP address is IP" to which the rewriter responds "Sure thing; the opaque handle corresponding to EID is OIP". These messages are exchanged via Click's `/proc` file system interface. *doad* is written in C++ and uses the SFS libasync library [18].

5.1.2 Packet Rewriter

The rewriter maintains a one-to-one mapping between OIPs and EIDs; some of the mappings are in-

[5]The network-to-host direction is opposite; both directions are discussed in more detail below.

serted by *doad*, described above, and others are dynamically created by the rewriter. Each EID is also associated with a globally reachable IP address. The mapping is invoked both in the host-to-network and network-to-host directions. Whereas the host-to-network direction maps OIPs in the packet to actual EIDs and associated IP addresses, in the network-to-host direction, the rewriter maps the source and destination EIDs to OIPs before handing the rewritten packet up to the kernel. In the network-to-host direction, if the packet rewriter sees a source EID it does not recognize, it first allocates a new OIP and then inserts EID (and associated IP address, which is the source IP address of the packet) and OIP together in a new mapping entry.

The mechanics of rewriting are as follows: in the host-to-network (network-to-host) direction, the rewriter changes the source and destination IP addresses from (to) OIPs; inserts (removes) a DOA header; changes the protocol field in the IP header to (from) 203; recalculates the IP checksum; and, if the packet is TCP or UDP, recalculates the pseudo-headers (which, for DOA packets, are taken over the IP header and the DOA header).

5.2 NAT Implementation under DOA

We used Click! to implement a DOA NAT, as described in Section 4.1. The NAT module maintains a simple lookup table, rewrites IP addresses and recomputes TCP checksums. A DOA NAT's lookup table is populated as the end-hosts behind the NAT boot up and run our implementation of DOA-RIP. The same protocol allows an end-host to discover its own external IP address, which is equal to the external IP address of the outermost NAT on its path to the core. After DOA-RIP completes, the end-host can advertise its location in the DHT: it *put*s a mapping between its EID and its external IP address.

5.3 Support for Legacy Applications

Legacy applications can use DOA with no recompilation, using techniques like those in [13]. Potential human users of this feature must be working on DOA-enabled hosts and must set their DNS resolver to our DOA-aware DNS proxy, *doadns*, which must be running on the local host. When a DNS query for, *e.g.*, `f12a.DOA` arrives—as happens when a human types `ssh f12a.doa`—*doadns* sends a `GetHostByEID` RPC to *doad* and returns the resulting opaque handle in an A-record. The legacy application treats the opaque handle as an IP address, so the application's traffic is steered to the DOA data path.

6 Related Work

Saltzer [23] was one among many [11, 17, 25] who made fine distinctions between network identifiers; the most common, and least practiced, of these distinctions is between a host's identifier and its address (see [16] for comprehensive discussion of this topic).

The arguments that these host identifiers should be flat and that a DHT should be used as the resolution mechanism are articulated in a recent proposal [1]. This proposal is an unabashed heist of insights and mechanisms from many places, including HIP [19–21] and SFR [31]. The HIP project, wishing to separate identity and location, advocates for a set of host identifiers with cryptographic properties; the SFR project articulates the case for flat names.

Today, the closest thing to a persistent host identifier in the presence of renumbering is a Dynamic DNS name (see [7] for an example of a commercial provider of a Dynamic DNS service). Dynamic DNS ensures that a given domain name always resolves to the given host's current IP address. Dynamic DNS has the same limitations as standard DNS for our purposes: because the DNS names are not inside the packet, middleboxes cannot use the names to tell for which host the packet is actually destined.

UIP [9] seeks to interconnect heterogenous networks by assigning each host a persistent and flat identifier with cryptographic properties. UIP uses a routing algorithm inspired by DHTs to route packets based on the identifier, and participating hosts route packets in UIP space for each other. UIP contrasts with our approach in that we resolve host identifiers to IP addresses all at once and then use IP routing to send the packet. Also, our goal is to enhance the functions of the middleboxes whereas UIP tries to make them transparent by using an overlay. Peernet [8] is another peer-to-peer inspired project that seeks to decouple identity and location. Its primary focus is the wireless realm.

The *i3* [27] project proposes a DHT-based infrastructure to decouple sending and receiving; the authors' intend to separate location and identity. The

vision of *i3* is that packets would contain flat identifiers representing hosts and that the act of sending to a host (represented by a flat identifier) would consist of injecting the packet into a DHT. The DHT node responsible for the flat identifier would receive the packet and then forward it to the ultimate destination. Whereas *i3* uses the DHT as a sending mechanism, we use it as a lookup service.

IPNL [12] shares many of our motivations. This work is intended to make renumbering easier, create separate end-host identifiers, and leave the core IPv4 routing infrastructure untouched. Under IPNL, the end-host identifiers are domain names, though the authors acknowledge that a flat, cryptographically strong identifier, as in HIP, may be preferable for security reasons.

P6P [30, 33] proposes a DHT-based infrastructure as a way to deploy IPv6: sites send IPv6 packets to their gateway DHT node, which treats the IPv6 destination address as a flat identifier, uses this identifier to look up the IPv4 address of a counterpart DHT gateway, and then sends the packet over traditional IPv4 to this counterpart, where the encapsulation is inverted and the packet is delivered to its destination. P6P shares many of our motivations but does not give hosts persistent names (if a site changes ISPs, all of the identifiers at the site change).

There are an increasing number of proposals for radically new network architectures. These include earlier proposals like PIP [10], IPv6 [6], Dynamic Networks [22], Active Networks [28], Nimrod [4], and more recent proposals like Smart Packets [24], Network Pointers [29], Role-Based Network Architecture [2], and Ephemeral State Processing [3]. Each of these proposals is an *ab initio* design that would (in its full glory) require significant modifications to all network elements, not just hosts.

Finally, FARA [5] presents a novel organization of network architecture concepts. While many of its goals are similar to ours, one directly conflicts: FARA deliberately avoids creating new global namespaces whereas we strongly advocate creating one new flat global namespace.

7 Discussion

DOA's fate will be decided by the functionality it simplifies and the applications it facilitates. However, we do not mean to say that DOA, by itself, directly enables any one application. Rather, DOA's benefit is improved architectural coherence. This coherence results from two aspects of DOA: first, that DOA is a platform (and is thus "built exactly once") and second that DOA is a layer of indirection. We believe these two aspects can foster innovative, yet previously impractical, technology.

We have taken a small step toward validating this belief: We illustrated here that DOA exports an architectural primitive (host identities contained in packets) that one can use to punch holes through NATs automatically, thereby allowing "previously impractical" technology. Below we speculate about other uses of the DOA architecture.

Packet washing. One type of middlebox not previously discussed in this paper is the *packet washer*: a machine that inspects packets on the way to a host and ensures that packet contents conform to the user's specification of "safe". For example, a firewall shields machines behind it from certain classes of traffic.

Today, the current network architecture requires these boxes to exist at network choke-points so that all packets destined for an end-host are forced through the intermediary. NATs and firewalls are often co-located for this reason. We argue that the robust notion of host identity provided by DOA, along with the ability to express to the network an *identity-independent location*, allows these packet-washers to exist anywhere in the Internet.

A host could use the DHT to map its EID to the IP address of the packet washer and also register with the packet washer each time its location changes. All packets destined for the host would then be automatically routed through the packet washer, which could demultiplex incoming packets based on the EID, inspect the packets, and then forward them to the ultimate destination. The host could then check and verify (perhaps with a signature or other token contained in packets) that a given packet originated from the packet washer.

DOA's abstraction of network location means that packet washers would not require topological proximity to their clients and, moreover, that packet washers could associate policies with host *identities*. This separation of function and location creates the opportunity for third-parties to provide packet washing as a service, perhaps benefiting from the

economies of scale for costs like hardware maintenance and virus definition updates.

DDoS remediation. An end-host using a packet washer would still be vulnerable to a distributed denial of service (DDoS) attack that overwhelmed the end-host's access link. However, we conjecture[6] that it is possible to defend against such attacks with the primitives given by DOA in conjunction with new pieces of infrastructure that are perhaps supplied by ISPs as a service. We might be able to borrow heavily from a recent proposal [32] for DDoS alleviation based on per-packet capabilities.

Other intermediaries. In addition to the packet washer discussed above, we imagine DOA could be useful for other types of intermediaries. We briefly give two examples.

First, consider a load balancer for Web servers. This load balancer accepts requests destined for a canonical EID that is associated with a given Web site. The load balancer distributes load by forwarding requests to any of several equivalent Web servers, as do today's load balancers. The primitive supplied by DOA, however, allows this machine to exist at a location other than a network choke-point, and the machines behind the load balancer to be globally reachable via their distinct EIDs (*e.g.*, for debugging and maintenance).

Second, DOA could simplify the logistics of conducting network measurement studies by taking all packets addressed from and to the EIDs of a consenting group of machines and then routing this traffic through a statistics-collection box for an arbitrary amount of time.

Although DOA's mechanism—a robust set of host identifiers that are carried in packets—was originally proposed to address mobility and multihoming problems, the NAT scenario that occupies much of this paper, and the examples just discussed, suggest that this same mechanism is actually a powerful architectural primitive for enhancing middleboxes. We hope that over time, many benefits will arise from this primitive. Discovering, enumerating, and fully flushing out these new possibilities is our future work.

[6] Whereas much of this paper is informed conjecture, this one is baseless.

References

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

[2] 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.

[3] K. L. Calvert, J. Griffioen, and S. Wen. Lightweight network support for scalable end-to-end services. In *ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

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

[5] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the addressing architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Karlsruhe, Germany, Aug. 2003.

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

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

[8] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. PeerNet: Pushing peer-to-peer down the stack. In *2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, Mar. 2003.

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

[10] P. Francis. A near-term architecture for deploying PIP. *IEEE Network*, 7(6):30–27, 1993.

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

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

[13] J. Kannan, K. Lakshminarayanan, I. Stoica, A. Kubota, and K. Wehrle. Supporting Legacy Applications over *i3*. Under submission to OSDI 2004. `http://i3.cs.berkeley.edu`.

[14] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring adoption of DHTs with OpenHash, a public DHT service. In *Proc. of the 3rd IPTPS*, February 2004.

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

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

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

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

[19] R. Moskowitz and P. Nikander. Host identity protocol architecture, Sep 2003. draft-moskowitz-hip-arch-05,

IETF draft (Work in Progress).

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

[21] P. Nikander, J. Ylitalo, and J. Wall. Integrating security, mobility, and multi-homing in a HIP way. In *Network and Distributed Systems Security Symposium (NDSS '03)*, pages 87–99, San Diego, CA, Feb 2003.

[22] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[23] 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.

[24] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart packets: applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, Feb. 2000.

[25] J. F. Shoch. Inter-network naming, addressing, and routing. In *17th IEEE Computer Society Conference (COMPCON '78)*, pages 72–79, Washington, DC, 1978.

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

[27] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proc. ACM SIGCOMM '02*, Aug. 2002.

[28] D. L. Tennenhouse, J. M. Smith, D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[29] C. Tschudin and R. Gold. Network Pointers. In *1st ACM Workshop on Hot Topics in Networks*, Princeton, NJ, Oct. 2002.

[30] R. van Renesse and L. Zhou. P6P: A peer-to-peer approach to Internet infrastructure. In *3rd International Workshop on Peer-to-Peer Systems*, San Diego, CA, Mar. 2004.

[31] M. Walfi sh, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, Mar. 2004.

[32] A. Yaar, D. Song, and A. Perrig. An endhost capability mechanism to mitigate DDoS fboding attacks. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[33] L. Zhou, R. van Renesse, and M. Marsh. Implementing IPv6 as a peer-to-peer overlay network. In *Workshop on Reliable Peer-to-Peer Distributed Systems, 21st IEEE Symposium on Reliable Distributed Systems (SRDS '02)*, Suita, Japan, Oct. 2002.