A Data-Oriented (and Beyond) Network Architecture

Teemu Koponen[*†]
tkoponen@iki.fi

Mohit Chawla[‡]
mchawla@cs.berkeley.edu

Byung-Gon Chun[‡]
bgchun@cs.berkeley.edu

Andrey Ermolinskiy[‡]
andreye@cs.berkeley.edu

Kye Hyun Kim[‡]
kyekim@cs.berkeley.edu

Scott Shenker[*‡]
shenker@icsi.berkeley.edu

Ion Stoica[‡]
istoica@cs.berkeley.edu

ABSTRACT

The Internet has evolved greatly from its original incarnation. For instance, the vast majority of current Internet usage is data retrieval and service access, whereas the architecture was designed around host-to-host applications such as telnet and ftp. Moreover, the original Internet was a purely transparent carrier of packets, but now the various network stakeholders use middleboxes to improve security and accelerate applications. To adapt to these changes, we propose the Data-Oriented Network Architecture (DONA), which involves a clean-slate redesign of Internet naming and name resolution.

Categories and Subject Descriptors

C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design

Keywords

Naming, Internet architecture, name resolution, data, middleboxes

1 Introduction

The DNS name resolution system is a fundamental part of today's Internet, underlying almost all Internet usage. However, the DNS was developed rather late in the Internet's evolution, after many basic pieces of the architecture were in place. For instance, TCP sessions were already bound to IP addresses and the Berkeley Socket API referred to addresses, not names; frozen design decisions, such as these, limited the extent to which DNS names (or any other naming system) could permeate the architecture. As a result, the current role of naming in the architecture is more an accident of history than the result of principled architectural design. In this proposal, we take a "clean-slate" look at naming and name resolution.

The test of any architecture is whether it gracefully accommodates a wide spectrum of potential uses (and can withstand potential abuses), both those we encounter in the present and those we anticipate for the future. However, to motivate our design, we first focus more narrowly on one particular issue; the shift in usage from host-centric to data-centric applications.

The first Internet applications, such as file transfer and remote login, focused strictly on host-to-host communication: The user explicitly directed the source to communicate with another host, and the network's only role was to carry packets to the destination address in the packet header. The Internet architecture was built around this host-to-host model and, as a result, the architecture is well-suited for communication between pairs of stationary hosts.

Today, however, the vast majority of Internet usage is data retrieval and service access, where the user cares about content and is oblivious to location. That is, the user knows that she wants headlines from CNN, or videos from YouTube, or access to her bank account, but does not know or care on which machine the desired data or service resides. The current architecture can support data/service access, as is obvious from its prevalence on the Internet, but it does not fit comfortably within the host-to-host model. For instance, consider the following three user-relevant issues:

- **Persistence:** once given a name for some data or service, the user would like that name to remain valid as long as the underlying data or service is available. There should be no equivalent of today's "broken links" when data is moved to another site. Today, HTTP redirect and dynamic DNS are used to minimize this problem, but they are not sufficient answers. For instance, neither works if the data changes administrative domains, unless the operator of the previous domain provides perpetual support.

- **Availability:** data and services should have high availability, in terms of both reliability and low-latency. Availability is usually provided by replication at endpoints, and the network's role is to allow user requests to find nearby copies. The first large-scale solution to this was deployed by Akamai, using intelligent DNS servers and URL rewriting. More recently, P2P mechanisms like BitTorrent have become prevalent. While the success of these mechanisms is undeniable, it is not at all clear that such a fundamental

[*]International Computer Science Institute (ICSI)
[†]Helsinki Institute for Information Technology (HIIT)
[‡]UC Berkeley, Computer Science Division

requirement, availability, should rely on a set of *ad hoc* and application-specific mechanisms.

- **Authenticity:** users would like to know that the data came from the appropriate source, rather than from some spoofing adversary. Today this requires a PKI to provide users with the public key of the provider. Moreover, authenticity today is typically achieved by securing the channel to the source, rather than explicitly authenticating the data.

Thus, several of the most natural features one would want for service access and data retrieval — persistence, availability, and authentication — are made unnecessarily hard by the current host-to-host model of the Internet, often requiring awkward or expensive work-arounds. Given this discordance between historical design (host-oriented) and current usage (data-oriented), we ask: what would the architecture look like if we built it around service and data access?

Somewhat surprisingly, our research suggests that most of the necessary changes reside in how Internet names are structured and resolved. We propose replacing DNS names with flat, self-certifying names, and replacing DNS name resolution with a name-based anycast primitive that lives above the IP layer. We call the resulting design the Data-Oriented Network Architecture (DONA).

DONA improves data retrieval and service access by providing stronger and more architecturally coherent support for persistence, availability, and authentication. It can also be extended to provide support for caching and RSS-like updates. However, DONA's impact is not limited to data and service access; we use these applications as motivating examples because they force us to think differently about some fundamental issues, but most of these issues are not particular to data/service access. As a result, as we describe below, DONA's overall design has architectural implications that range far beyond data/service access.

DONA's name-based anycast primitive is useful for many kinds of resource discovery; for instance, it can provide the basic primitives underlying SIP, support host mobility and multihoming, and establish forwarding state for interdomain multicast. Placing anycast at the naming layer, rather than at the IP layer, allows us to design for functionality rather than be limited by concerns about scalability, since the mechanisms need not operate at link speed.

There is another issue where historical design is at odds with current usage. The original Internet architecture, following the end-to-end principle, intended the network to be a purely transparent carrier of packets. Today, however, the various network stakeholders (such as enterprises) use middleboxes to improve security (*e.g.*, firewalls, proxies) and accelerate applications (*e.g.*, caches) [3]. Because DONA's anycast name resolution process follows essentially the same administrative path as the ensuing data packets (we will address the subtleties in this statement later), DONA can treat the stakeholders along the path as relevant Internet actors. This allows DONA to provide clean support for network-imposed middleboxes. This isn't a repudiation of the end-to-end principle, in that functionality is still provided at the ends; it is merely a recognition that operators should have, at their disposal, architecturally coherent mechanisms to control how and what traffic traverses their network.

More recently, there has been much hand-wringing about the scalability of routing in the current addressing paradigm [27]. DONA's anycast primitive provides a discovery mechanism that lives above the IP layer; as we will later describe, this enables the use of path-labels rather than global addresses, an approach that results in tiny interdomain routing tables.

At a more speculative level, DONA represents a partial shift away from sender-based primitives to a more receiver-based approach. One of our future research tasks is to explore how far we can go in this direction, and what this might mean for a future Internet.

These architectural implications encouraged us that DONA is not merely restricted to data and service access (which, by itself, is significant as it is by far the dominant usage on the current Internet), but rather facilitates improvements along many dimensions. However, there are many other issues, not discussed here, that demand attention: the Internet still needs better security (particularly against DoS and malicious/misconfigured routers), better manageability, better usability, and many other properties. We aren't proposing DONA as a solution to these problems; in fact, we think DONA is largely orthogonal to them. We hope to eventually incorporate work on these problems within a larger framework that also includes DONA.

The next section presents DONA's basic design and Section 3 describes how this design supports such tasks as server selection, mobility, multihoming, session initiation, and interdomain multicast state establishment. Section 4 discusses how DONA's infrastructure could be extended to support more advanced functionality, such as content delivery, delay-tolerant networking, and a variety of administrative access policies (including middlebox insertion).

Section 5 discusses our prototype implementation and Section 6 addresses the crucial question of DONA's feasibility. The name-based anycast primitive will require routing on a very large namespace, but it need only be done at name resolution speeds, not line speeds; we present various estimates indicating that DONA is within reach of today's technology.

We delay our discussion of related work until Section 7, in order to have enough context to make the necessary connections. For now, we merely note that almost every aspect of our design is (proudly) stolen from elsewhere, most notably from TRIAD [19], HIP [28], and SFS [25]. It is the synthesis of these various ideas into a coherent architecture that we claim as our contribution.

The paper ends, in Section 8, with some speculations on DONA's broader architectural implications. In particular, we discuss the possibility of basing the interface offered to applications on DONA's name-based anycast primitive.

2 Basic Design

DONA involves a major redesign of Internet naming. In this section we first motivate these changes and then present DONA's naming structure and name resolution mechanism. This is followed by a brief discussion of security and addressing issues. For lack of space, many details are omitted.

2.1 Motivation

We start with the problem of service and data access and ask how we might easily achieve persistence, availability, and authentication, basic tasks which today are (sometimes badly) handled by external *ad hoc* mechanisms. In DONA, we propose a strict separation of concerns between naming and name resolution in handling these tasks: names handle persistence and authenticity, while name resolution handles availability.

To provide persistence and authenticity, we use flat, self-certifying names [25,28]. This form of naming is, by now, a standard technique. As we review shortly in Section 2.2, such names will remain invariant and enable easy authentication. The use of flat names makes informal identification harder (since you can't remember your friend's 128-bit identifier), but it makes formal authentication easier.

High availability means that when a user requests data by name, she receives the data quickly and reliably. To provide availability, the name resolution process should (a) guide requests to nearby copies of the data, and (b) avoid failed or overloaded servers. The question, then, is how to build such a name resolution process.

There are two main resolution paradigms in the literature. The first is what we use today: lookup-by-name in a distributed database, which returns the location (IP address) of a nearby copy. This database must maintain locations of all the copies, identify the location of the requester, and then find a reasonably good match between the two. Akamai has pioneered the development of techniques to accomplish this, but it clearly requires significant mechanism to achieve.

The other possibility, most notably used in TRIAD [19], is to *route-by-name* to the closest copy. Routing protocols are designed to find shortest paths and route around failures, exactly the two tasks (a) and (b) we've assigned to name resolution. This led us to conclude that route-by-name, rather than look-up, was the most natural approach. We discuss our design for this in Section 2.3.

We note that there is one other issue — in addition to persistence, authenticity, and availability — that the user cares about, trustworthiness: users would like to know whether they are getting their information from a reliable source. We believe that this, like several other issues we discuss below, is best handled by mechanisms that are external to the architecture. Trust is so idiosyncratic and subjective that we don't believe any network architecture should mandate the mechanisms by which trust is established. Moreover, by remaining outside of the core architectural structures, external trust mechanisms can evolve along with changes in society and institutions in a way that a fixed architecture can't. Today, a variety of external mechanisms, ranging from Google to personal recommendations, are used to establish trust. We expect that new trust mechanisms, such as reputation systems and enhanced "webs-of-trust", will be developed in the future, but they will (and should) lie outside the confines of the architecture.

2.2 Naming

DONA names are organized around *principals*. Each principal is associated with a public-private key pair, and each datum or service or any other named entity (host, domain, etc.) is associated with a principal. Names are of the form P:L where P is the cryptographic hash of the principal's public key and L is a label chosen by the principal, who ensures that these names are unique. The granularity of naming is left up to principals; a principal might choose to just name her web site, or name her web site and each page within it, or name at a finer granularity (such as naming each individual photo or publication).

Principals are considered to *own* their data, in the sense that only hosts authorized by the principal P can offer to serve (*i.e.*, provide access to) entities with names of the form P:L. Each datum comes with metadata including the principal's public key and the principal's signature of the data; thus, when we speak of a client retrieving data we mean it has received the triplet <data, public key, signature> (along with perhaps other metadata).[1] In such a scheme, requesting clients rely on the principal's signature to ensure the data's integrity.

These names are application-independent and globally unique (and can refer to anything, not just data or services). They are also *self-certifying* in the following sense [25,28]: When a client asks for a piece of data with name P:L and receives the triplet <data, public key, signature>, it can immediately verify that the data did indeed come from the principal by checking that the public key hashes to P, and that this key also generated the signature. This satisfies the need for authentication; persistence follows from the fact that the names don't refer to location, and thus the data can be hosted anywhere.

With a slight alteration, these basic ideas can be naturally applied to immutable data: here, the label L is the cryptographic hash of the contents of the data and the principal P is the purveyor of the data, not the owner; for instance, the purveyor could be the hosting CDN. Since the client need not rely on a principal to ensure the integrity of the data (the hash over the contents ensures this), the only role of the principal is to ensure data delivery. Since different CDNs may have different degrees of reliability or coverage, clients seeking immutable data might specifically request it from a particular purveyor.

Note that there is a difference between the administrative structure of the hosting machines, and the nature of the principal. A person's web page would be associated with their own public key. The web page might initially be hosted at their university or company or paid hosting service, but this is not reflected in the name; instead, the owner (as we discuss below) would authorize (with some reasonable TTL) this entity to host their web page (and this authorization could be applied to only a specific datum, or to a portion of the principal's data, or to all the principal's data). If the person decided to move the page (for instance, if they changed employers), then they would authorize a new entity to host their page (and let the old authorization expire). The name of the data would not change, even though the entity hosting the data (or service) did change.

The biggest challenge to making such flat names work is making sure they *resolve* to the appropriate locations, which is what we discuss below. But there are usage questions that must also be addressed (though we are not the first to propose names of this form, and several of these issues have been discussed elsewhere, such as in [37]).

For instance, one usage question is: how will users learn these flat, long, and user-unfriendly names? We expect that users will learn these flat names through a variety of external mechanisms that the user trusts (to varying degrees), such as search engines, private communication, recommender services, and the like. Users won't, of course, remember the flat names directly, but will have their own private namespace of human-readable names, which map onto these global and flat names (as in [14]). While such flat names are harder to use than today's DNS names, they offer the advantage that the mappings between private human-readable names and flat names will be free to reflect evolving social structures rather than being tied, as DNS names are, to a fixed administrative structure.

Also, DONA does not provide a *reverse-lookup* mechanism for names, like DNS, where one can map an IP address to a name. The basic use of reverse-lookup is to tie a user-unreadable label (in this case an IP address) to an user-meaningful identity (in this case, a DNS name). We think the analogous service in DONA would allow a user to map a principal's key (which is human-unreadable) to a trust-chain between the principal's key and his own that would help the user understand the identity of the principal.

2.3 Name Resolution

We now turn from discussing the nature of DONA's names to presenting how DONA translates these names into locations. Our goal in this design is to achieve high availability, by finding close-by copies and avoiding failures.

As discussed earlier, DONA uses the *route-by-name* paradigm for name resolution. Rather than use DNS servers, DONA will rely on a new class of network entities called *resolution handlers* (RHs). Name resolution is accomplished through the use of two basic primitives: FIND(P:L) and REGISTER(P:L). A client issues a

[1]The signature might be signed with a secondary key, but accompanied by a certificate from the principal's key authorizing the secondary key.

FIND(P:L) packet to locate the object named P:L, and RHs route this request towards a nearby copy. REGISTER messages set up the state necessary for the RHs to route FINDs effectively.

Each domain or administrative entity will have one logical RH (but perhaps many physical incarnations); we will denote the RH associated with an administrative entity X by RH_X. RH_X is the provider/customer/peer (or, alternatively, parent/child/peer) of RH_Y if X is the provider/customer/peer of Y in terms of AS-level relationships. This RH structure can extend to finer granularity than ASes to reflect other organizational and social structures; for instance, there could be departmental RHs at universities and firms and, going even further, users could have their own local RHs which peer with those of their neighbors and friends. RHs use local policy (consistent with their domain's peering agreements) when processing REGISTERs and FINDs.

Each client knows the location of its local RH through some local configuration (much like they know about their local DNS server). Any machine authorized to serve a datum or service with name P:L sends a REGISTER(P:L) command to its local RH. Registrations can also take the form REGISTER(P:*) if the host is serving all data associated with the principal (or will forward incoming FIND packets to a local copy).

Each RH maintains a *registration table* that maps a name to both a next-hop RH and the distance to the copy (in terms of the number of RH hops, or some other metric). There is a separate entry for P:*, in addition to individual entries for the various P:L. RHs use longest-prefix matching; if a FIND for P:L arrives and there is an entry for P:* but not P:L, the RH uses the entry for P:*; when entries for both P:* and P:L exist, the RH uses the one for P:L. Only when the RH has neither P:* or P:L entries do we say that P:L does not have an entry in the registration table.

When a FIND(P:L) arrives, the forwarding rule is straightforward: if there is an entry in the registration table, the FIND is sent to the next-hop RH (and if there is more than one, the choice is based on the local policy and which entry is closest); otherwise, the RH forwards the FIND towards its parent (*i.e.*, its provider) using its local policy to choose among them if the RH is multi-homed. Thus, registration table misses are forwarded up the AS hierarchy in the hope of finding an entry (see Figure 1). In the case of immutable data, a FIND command can take the normal form FIND(P:L), or the special form FIND(*:L) which indicates that the client is willing to receive the (self-certified) data from any purveyor.

If RH_X receives a REGISTER from a child (*i.e.*, customer), it does not forward it onward unless no such record exists or the new REGISTER comes from a copy closer than the previous copy. If so, RH_X forwards the REGISTER to its parents and peers (after updating its registration table). If the REGISTER comes from a peer, the entry can be forwarded or not based on local policy (depending, for example, on whether the AS is willing to serve as a transit AS for content). By letting the forwarding of FINDs and REGISTERs be driven by the local policies, DONA can faithfully respect the basic interdomain policies as reflected in BGP. In addition, the forwarding of a REGISTER can be terminated at any point if dictated by some administrative policy (such as a corporate firewall).

REGISTER commands must be authenticated. The local RH issues a challenge with a nonce, which the client must sign with P's private key, or sign with some other key and provide a certificate from P empowering this other key to register this piece of data. When forwarding REGISTERs, the RH signs it so that the receiving RHs know that the data came from a trusted RH. These signatures are hop-by-hop and accumulated in a REGISTER along the path. In a similar manner, the RHs accumulate the distances; they append their distance/cost to the previous-hop RH before sending

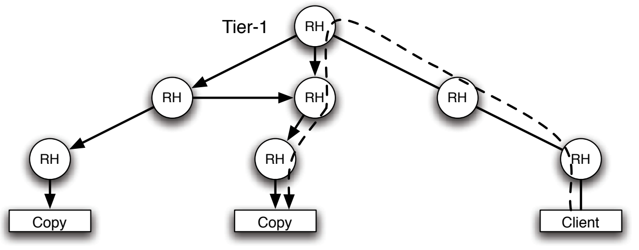

Figure 1: Registration state (solid arrows) in RHs after copies have registered themselves. RHs route client-issued FIND (dashed arrow) to a nearby copy.

register : the received REGISTER message, if any.
regs : all REGISTER messages for the name (P:L).
pref_reg: preferred REGISTER to disseminate to peers/parents, if any.
name : the name (P:L) event concerns.

```
regs ←load(name);
if register received then
    if duplicate or invalid signature then
        return;
    end
    set_timer_for_expiration(register);
else
    // A REGISTER expired...
end
foreach out in provider and peer links do
    pref_reg ←decision_process(out,regs);
    if pref_reg changed for out then
        msg ←new_message(pref_reg);
        add_intra_cost(out,msg);
        sign_message(private_key,msg);
        queue(out,msg);
    end
end
store(name, all changes);
```

Figure 2: Pseudo code for processing received and expired REGISTERs.

the REGISTER to next RH. REGISTER commands have a TTL and must be refreshed periodically. DONA also provides an UNREGISTER command so that clients can indicate that they are no longer serving some datum. Figure 2 shows the pseudo code for processing received and expired REGISTERs.

The FIND packet does not just resolve the name, it also initiates the transport exchange. The FIND packet takes the form as shown in Figure 3, where the DONA-related content is essentially inserted as a shim layer between the IP and transport headers. The name-based routing provided by DONA ensures that the packet reaches an appropriate destination. If the FIND request reaches a Tier-1 AS and doesn't find a record associated with that principal, then the Tier-1 RH returns an error message to the source of the FIND. If the FIND does locate a record, the responding server returns a standard transport-level response (the same as if the transport header had been received on a normal data packet, not on a FIND packet). To make this work, transport protocols should bind to names, not addresses, but otherwise do not need to change. Similarly, application protocols need only be modified to use names, not addresses, when calling

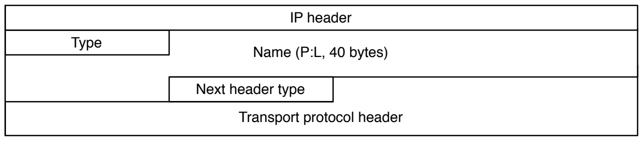

Figure 3: Protocol headers of a FIND packet. Type is to separate FINDs from their responses.

transport. In fact, many applications could be simplified when implemented on top of DONA. Using HTTP as an example, we note that the only essential information in an HTTP initiation is the URL and header information (such as language, etc.); the URL is not needed, since the data is already named in a lower layer, and if each variation of the data (such as language) is given a separate name then the header information is also superfluous.

The packet exchanges that occur after a FIND has been received are not handled by RHs (except, as we note in Section 4, when they serve as caches or other middleboxes), but instead are routed to the appropriate destination using standard IP routing and forwarding. To this extent, DONA does not require modifications of the IP infrastructure.

2.4 Security Issues

There are a variety of security issues that must be addressed, some by DONA itself and some by underlying or external mechanisms.

For bandwidth denial-of-service attacks, we assume that there are IP-level mechanisms that can restrain unwanted packet streams that are overwhelming an RH, server, or client. For resource exhaustion attacks against RHs, DONA relies on contractual limits providers place on customers for the number of FINDs and REGISTERs they can submit per time period. RHs may additionally impose other rate-limiting techniques such as cryptographic puzzles.

We assume that as part of establishing customer/provider/ peering relationships, peering RHs have securely exchanged their public keys, so RHs can always ensure that they are receiving packets from the appropriate RH. However, a malicious RH can still cause damage. For instance, a malicious RH can refuse to forward REGISTERs and FINDs; this is a failure of the AS, in much the same way an AS could fail to forward packets, and presumably commercial pressures would reduce this form of misbehavior. More subtly, a malicious RH could forward REGISTERs overheard from other RHs. To minimize this risk, when RHs forward a REGISTER they include the next-hop's public key (or its cryptographic hash).

The worst a malicious RH can do is deny a client service (since cryptographic measures allow the client to authenticate the data). In Section 3 we discuss ways in which clients can request access to other copies (*i.e.*, not just the closest one); this will allow a client to avoid misbehaving RHs, unless the misbehaving RH lies on the path to all copies of that particular item.

In all cases, though, RHs are commercially related to the clients they are serving; they are paid (perhaps recursively) by either the client or the server. Thus, DONA is not relying on the cooperation of arbitrary entities, but is relying on the nature of a commercially provided service. Thus, while the design should be reasonably secure against misconfigured or subverted RHs, we anticipate that any such problems would be detected and corrected by the provider.

In such a key-centric design, the greatest fear is key compromise. There is no remedy except for providing effective means for key revocation. DONA does not, itself, provide a key revocation mechanism. However, there are a variety of mechanisms one could use for this, such as third-parties (*e.g.*, Google) providing databases of revocation lists. Each revocation is cryptographically proven, so they cannot be faked; thus the database need merely provide access to the data, not vouch for its correctness.

DONA could itself provide a useful substrate for revocation lists and online key status query protocols. That is, key revocations related to a principal P (both of P's private key, and of any secondary keys P has used) could be stored under P:L for some special reserved name L, and if an RH finds any entry corresponding to that name it immediately returns notification of a key compromise.[2] Thus, if a client wanted to check a key related to P it could issue a FIND(P:L) for this special value of L. As we describe in Section 4, DONA also supports update functionality, so a client could subscribe to be notified of any such revocation.

Finally, principals and replicas may ensure that their key is not already in use by doing a FIND(P:*) on a freshly generated key P (using DONA's name resolution).

2.5 Internet Addressing

The DONA design, as just described, could function over the current IP layer, with its present form of addressing and routing. However, many think the current Internet addressing scheme is facing a looming crisis, as the increasing demand for multihoming threatens to explode routing tables [27]. Even aside from this speculative threat, the current addressing paradigm requires a delicate balance between scalability (*e.g.*, aggregation) and flexibility (*e.g.*, multihoming, policy routing) that isn't always easy to achieve.

DONA's name-based anycast primitive can remove much of the pressure on the lower-level addressing structure by providing a separate mechanism for path discovery. In particular, DONA could enable IP to use path-labels (as in [22]) rather than globally routable addresses. In what follows, we refer to the client as the source of the FIND and the server as the node that responds to the FIND (presumably a node that generated the REGISTER, or a caching RH as discussed later in Section 4). Moreover, each host has a domain-specific address; that is, for each domain within which it is homed, that domain associates an address to that host, and that address has no meaning outside of that domain.

In this approach, when a client sends a FIND, its source address is originally just its domain-specific address. As the FIND is forwarded from client to server, next-hop domain path instructions are appended to this source address. Each such instruction has purely local meaning; for instance, as the FIND passes from domain A to domain B, an annotation is added to the path instruction that tells A that the next-hop domain is B and, vice-versa, tells B that, in the reverse direction, the next hop is A. This instruction need only be understood by the two connected domains A and B. When the FIND arrives at the server, the server appends its domain-specific address to the path description. It can then reverse these path instructions and use them for its response to the client (since reversing the order just gives the path in the opposite direction). Similarly, when the server's packets arrive at the client, the client can reverse the path in order to send packets to the server.

Because these per-hop path instructions only need to distinguish between the various next-hop domains, they can be quite short (say, on the order of a few bytes), so the total path instruction would be quite short. More importantly, the interdomain routing tables would be extremely small (and quite static); merely enough to translate these per-hop instructions into a next-hop AS. Note that these path-instructions would not have global meaning, since if a source in a different domain used this path, the domain-specific next-hop instructions would not necessarily lead to the desired

[2]Note that once any key revocation entry has been registered for that principal, there is nothing the key compromiser can do to cause its removal from the registration tables.

destination. Thus, in this design, there would be no globally meaningful addresses and the DONA FIND/REGISTER primitives would be required to establish end-to-end connectivity.

This approach would require the endpoints to detect AS-level path failures and to resend a FIND in that case. This is a substantial extra burden on connections, but it is the tradeoff for doing path-discovery above the IP layer. Also, while this design might superficially resemble other connection-oriented designs, with the FIND playing the role of a connection-establishment, an important distinction is that in DONA there is no per-flow state in the network.

This approach would produce symmetric AS-level paths, because the path of the FINDs lay down path instructions which guide the reverse path. There are other possibilities for how policy routing could be handled in this case, ranging from allowing asymmetric routes but requiring a FIND sent in the opposite direction (like many capability mechanisms [41,43]), or giving policy control going from the client to the core (and from the core to the client) to the providers near the client, and giving policy control going from the core to the server (and from the server to the core) to the providers near the server; this would preserve AS-level path symmetry but distribute the control differently among the ASes. Lastly, one could adopt a NIRA-like [42] approach to providing endpoint-control. We do not explore these options here.

3 Using DONA's Basic Functionality

DONA's name resolution process is essentially a name-based anycast service: a FIND(P:L) request is routed (by RHs) to a nearby RH at which the name P:L is registered. We now discuss a few ways in which this basic primitive might be used (and later, in Section 4, we discuss how RHs could be extended to provide greater functionality). We present only a very high-level description of the examples below, omitting (due to space) many protocol details.

Server Selection: The most basic use of DONA's anycast primitive is to select among several possible servers (*e.g.*, for content distribution, or replicated service access). Each server (or datacenter) authorized by a principal P to host a service or datum named P:L simply registers P:L at their local RH. DONA routes any FIND(P:L) to the closest such server (closest according to the DONA routing metric). Note that these servers could be from one or many different CDN networks; the name P:L remains the same no matter which server is providing the data.

P2P applications could also employ this primitive, and would probably use immutable names where the principal would identify the particular P2P infrastructure (these may have different degrees of reliability and coverage). Systems like BitTorrent that break files into chunks are also supported; the name of the file (immutable data) is mapped to an index of the chunk names (also immutable), and the client can then separately request each chunk. If it so desires, the client can then offer to serve the content, registering both the chunks and the index. We will see later in Section 4 how increased functionality in the RHs can further improve the CDN-like properties of DONA.

Mobility and Multihoming: A roaming host can first unregister from one location and then re-register at its new location. Subsequent FINDs will be routed to the new location as soon as the new registrations have installed the necessary state.[3] Well-known techniques below the transport layer [28], at the transport layer [17], or at the session-layer [32] can be used to mask mobility from higher layers. Multihoming is similarly straightforward: a multihomed host registers with each local RH and a multihomed domain forwards its REGISTERs to each provider. This allows FINDs, and thus the resulting data connections, to make use of multiple paths. While the above discussion implicitly assumed current IP addressing, and no need for symmetric AS-level paths, similar multihoming could be achieved with the addressing described in Section 2.5.

Session Initiation: Rendezvous mechanisms are the core of application-layer session initiation and presence protocols. Consider, for instance, the Session Initiation Protocol (SIP) [31]. A SIP user agent begins by sending a SIP INVITE message. The SIP proxy infrastructure then routes the INVITE message to the current location of the remote agent, which responds to begin the session negotiation. To keep their location up-to-date in the proxy infrastructure, SIP user agents register their current location with registrars (often co-located with SIP proxies). This process maps directly onto DONA's basic primitives. SIP INVITE messages translate to FINDs and the SIP and DONA REGISTER messages play the same role. Thus, SIP's rendezvous functionality can be provided by DONA and SIP's negotiation functionality could be implemented directly on top of DONA's REGISTER and FIND. In Section 4, we show how DONA enables the network to impose middleboxes; networks could impose (stateful and stateless) SIP proxies to enrich SIP services, control service access, and to protect themselves and their customers just as they do today (*e.g.*, SIP providers often deploy session border control boxes to rewrite SIP messages to hide their internal network topologies). Of course, after DONA enables the discovery, the data transfer between the two endpoints occurs over IP.

Multicast State Establishment: It has been a long struggle to define a simple and scalable interdomain multicast protocol. We now show how DONA could be used to establish this interdomain multicast state in a straightforward manner (similar to how this is done in [29]). In this design, DONA's anycast primitive provides the tree discovery function, allowing a domain's border router that has local members in a multicast group G to discover and establish connectivity with other domains that have members in the group. We assume that each domain runs some intradomain multicast protocol. Each multicast group has a name of the form P:G, where the principal is the originator of the group. Such a structure makes it easy to keep group names unique. When a new node in a particular domain joins P:G, the domain's border router R_{new} issues a FIND(P:G) packet which DONA routes to the nearest router R that also has local members in P:G, if one exists. Upon receiving the FIND(P:G) packet, R attaches R_{new} to the overlay topology for P:G as its neighbor and the two routers add appropriate entries to their neighbor tables for P:G.[4] To complete the join operation, R_{new} sends a REGISTER(P:G) command, announcing its membership and willingness to serve as an attachment point for other incoming group members. This construction ensures that the resulting overlay topology remains acyclic at all times, thereby simplifying packet forwarding.

To transmit a multicast packet destined for a particular group P:G, the sender's border router R similarly issues a FIND packet to locate a nearby domain that belongs to the group and forwards the packet to that domain's border router, which in turn initiates the packet's dissemination. Note that if the sender's domain has one or more members in P:G then R is itself a member of the overlay topology for P:G and has the forwarding state necessary to initiate the dissemination.

[3] A host that suddenly loses its connectivity may have to wait until its previous registration's TTL expires before it can be sure that subsequent FINDs will find it.

[4] By overlay we mean that the two routers tunnel packets to each other, so the intervening routers need not have routing state for this multicast group.

find: the FIND packet received.

```
if unknown transport or application protocol then
    forward_to_next_hop(find);
else
    if content cached and still valid then
        terminate_protocols(find);
        send_content();
    else
        // Follow the caching policy
        if should be cached then
            terminate_protocols(find);
            new_find ←clone(find);
            change_src_address(new_find);
            initiate_protocols(new_find);
            send_to_next_hop(new_find);
            store_and_send_content();
        else
            forward_to_next_hop(find);
        end
    end
end
```

Figure 4: Pseudo code for caching logic.

4 Extending DONA

As we've just discussed, DONA's name-based anycast primitive provides support for a range of discovery-like tasks. Even restricted to this capability, we believe DONA would be a better naming foundation for the Internet than the current DNS system. In this section we discuss extensions to DONA that broaden its impact. These extensions involve adding functionality to the RHs, so they take a more active role in the architecture than just forwarding FINDs and REGISTERs .

4.1 Improving Content Delivery

The basic DONA design provides support for server selection, which is useful when accessing widely replicated data or services. However, DONA can provide better support for content delivery in three ways.

Caching: RHs can be extended to provide a universal (*i.e.*, general-purpose and always on-path) caching infrastructure. An RH that implements caching must first populate its cache. It can do so by changing the source IP address of an incoming FIND packet to be its own (and also source port number, if necessary) before forwarding the FIND to the next-hop RH. This ensures that the response to the FIND will traverse this RH, so the RH will receive the returning data and can install it in its cache. As per current practice, cacheable data items will be labeled with a TTL or other invalidation metadata that determines when data becomes stale. When a FIND arrives and there is a cache hit, the RH responds to the FIND's source IP address, returning appropriate transport responses which then will proceed into a standard application-level exchange. If the RH does not understand the transport or application-level protocol (as defined by the port in the transport header) for a particular FIND, it does not provide caching for that request. Figure 4 shows the pseudo code for the above logic and Figure 5 depicts the packet exchanges.

Subscriptions: Often clients would like to subscribe for updates, as in RSS. This can be easily accomplished by adding TTLs to FINDs. When the server responds to such a TTL'ed FIND, it notes whether and how long it will provide updates to the FIND. When a server

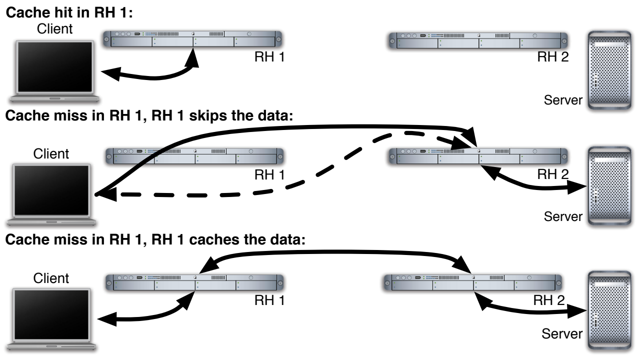
Cache hit in RH 1:
Client
RH 1
RH 2
Server
Cache miss in RH 1, RH 1 skips the data:
Client
RH 1
RH 2
Server
Cache miss in RH 1, RH 1 caches the data:
Client
RH 1
RH 2
Server

Figure 5: Three cases of processing a FIND packet when caching is supported: serve from the cache, skip (merely forward the FIND), and cache. RH 2 is a middlebox for all traffic.

updates a piece of content that has a pending TTL'ed FIND, it sends the update to the source of the FIND. RHs can assist in this process by caching TTL'ed FINDs, using the previously described technique of inserting their own address before forwarding the FIND. If a collection of RHs have cached the TTL'ed FINDs for a particular item, they form a distribution tree that provides scalable updates to that item. There is no explicit tree construction algorithm; it is formed via the normal routing of FINDs.

Avoiding Misbehaving and Overloaded Servers: In general, RHs will route FINDs to nearby copies of the data. However, some of these servers may be misbehaving (due to failures or malicious intent) in a way that is not visible to the RHs but which deprives the client of a valid copy of the data. To make sure that clients can still access the data even when their closest replica server is misbehaving, we allow the client to ask that its FIND be routed to a different server. In particular, we amend the REGISTER commands to keep track of the number of servers below a particular RH (this number is incremented as when a REGISTER is sent upwards in the RH graph), and we amend the FIND command to allow the client to request access to the k'th closest server, rather than the closest one. This may be particularly useful when accessing data from relatively unreliable P2P servers.

We can also augment DONA to allow overloaded servers to indicate they are overloaded, so the RHs can then direct excess load to other, less-loaded, servers. When an RH sends REGISTERs to a peer, it includes the distance to and load on the closest server and also, if the closest server is overloaded, the distance to the closest underloaded server; thus, at any time, an RH knows the next-hop towards the closest server with spare capacity. We have simulated the algorithm and find that it comes close to matching the optimal load distribution in several settings.

With these three changes, DONA could provide a combined caching, updating, and load-balancing infrastructure for content delivery. High-end commercial content providers may still require more advanced features and more careful monitoring than DONA can provide, but we expect even these high-end CDN services to augment DONA's content-delivery infrastructure rather than replace it entirely.

4.2 Delay-Tolerant Networking

There has been recent interest in developing architectures for networks where end-to-end connectivity may rarely exist, but sporadic hop-by-hop connectivity is often present (see *e.g.*, [6, 12]). These connectivity-challenged networks occur in many settings, ranging from large (interplanetary communication), to small (sensornets), to deep (oceanic communications). The key concept in designing a delay-tolerant network (DTN) architecture is message-level *custody transfer*; rather than the communication being end-to-end, there are intermediate elements that take custody of the message and then are responsible for making sure it is forwarded onward. In DONA, the RHs can act as these custody agents (or can deputize other hosts to act as custody agents). If an RH knows that connectivity to its neighboring RH is intermittent, it can choose to accept custody of the subsequent transfer in much the same way as it places itself on the return path for caching. Previous work on DTNs [12] has also embraced name-based routing. Though these two uses of name-based routing are realized somewhat differently, the similarity in approaches suggests that DONA might provide a reasonable substrate for DTN architectures.

4.3 Access Rules and Middleboxes

The original Internet architecture was built around the end-to-end principle, with intermediate routers playing no role other than forwarding packets to their destination. However, commercial and security pressures have created an Internet where layer-violating access policies and on-path middleboxes are common. There is no support for access rules and middleboxes in the current architecture; some maintain that this is a feature, not a bug, but we suggest that the presence of so many access rules and middleboxes is an indication of important unmet needs in the current architecture, and that any viable future design must address those needs.

One can view FINDs and REGISTERs as a general signaling mechanism; these are addressed at the IP layer to each RH along the path, and as such the RHs are not violating layering by inspecting them. Corporate networks may choose to not forward REGISTERs originating at internal RHs to external RHs if they don't want internal content available to the public. Similarly, such networks might also choose to not forward FINDs originating at external RHs to internal servers, or perhaps make the decision based on the application-port found in the FIND's transport header.

More generally, there are several natural policy decisions an RH can make upon receiving a FIND. The RH could deny the FIND, either failing silently or returning an error code. The RH could also ask the source of the FIND for authentication; for instance, it could ask for credentials proving that the user was indeed an employee. DONA itself would not standardize the format and nature of these credentials; DONA would merely provide a channel for authentication protocols to exchange these credentials.

An RH could also impose a middlebox, such as an application-specific proxy or a firewall, which is otherwise off-path. There has been previous work on incorporating endpoint-imposed middleboxes (see [2, 38]), where either the sender or receiver desires the packets to traverse a middlebox such as a firewall or transcoder. The middleboxes we consider here are network-imposed; the decision is being made by one or more of the networks carrying the packets, not the endpoints. We expect future networks to have a mixture of network-imposed and endpoint-imposed middleboxes, so we see the two approaches (DONA and [2, 38]) as complementary.

An example of an authentication request and a middlebox insertion is shown in Figure 6. Here, the FIND goes from client to RH1 to RH2, which forwards it to the firewall FW. The firewall asks the client for authentication, and the client responds; the firewall then forwards the client's FIND to the server, inserting its own IP address in the FIND. The server responds to the firewall, which then forwards the response to the client. Finally, the data flows between the client and server via the explicitly addressed firewall.

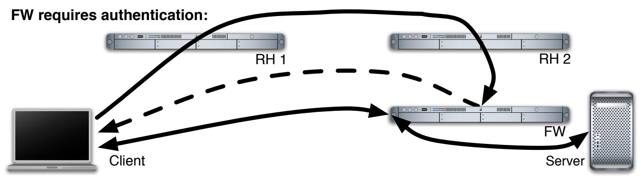

Figure 6: Server-side RH imposes a firewall middlebox which requires authentication (dashed arrow) before becoming a proxy for client-server communication.

When RHs play the role of caches, both in responding to FINDs and inserting themselves on the return path to receive the response, they are acting as middleboxes (as shown in Figure 5). RHs can similarly insert themselves as a middlebox for reasons other than caching, and, as just shown in the example in Figure 6, can also forward the FIND to some other box (*e.g.*, a firewall) that inserts itself on the path. This is done by inserting the IP address of the middlebox in the source address field in the FIND packet (and changing the transport source port, as well as updating checksums and TTL fields) before it is forwarded onwards, so all returning packets pass through the middlebox. This method of inserting middleboxes does not violate layering, as all packets intercepted by these middleboxes would have their IP address in the destination field.

Lastly, an RH could also provide the *capabilities* required to traverse the domain. We won't delve into the details here, but mechanisms similar to SIFF [41] and TVA [43] could be used (but on a per-domain basis, not per-router basis), with the capabilities inserted in the FIND as it traversed domains. There are two subtleties here. First, if more than one intervening domain wants to require capabilities, there must be a way of either concatenating the capabilities, or somehow sharing a fixed length capabilities field. The second subtlety is that using FINDs to gather the capabilities works if the FIND packet and the data packets being sent in both directions take the same AS-level (or administrative-level) path. This symmetry is required only for the networks that use capabilities. Today, restrictive access policies are typically only exerted at the edges, where the paths are likely to be symmetric. However, if one expects that capabilities might be used more generally, the path-labels discussed in Section 2.5 could be used, because they produce symmetric AS-level paths. In contrast to capabilities, middlebox insertion does not require symmetric AS-level paths, because the middlebox is explicitly addressed.

5 Implementation

To gain some experience with our design, we implemented a Linux-based prototype of an RH and deployed it on Planetlab. Implementing DONA helped us uncover many details that we had neglected, but it did not raise any fundamental issues. We think the far more interesting issue is large-scale feasibility, which we could not address in our small-scale prototype deployment; we address these scaling issues in Section 6.

The RH prototype is a stand-alone user-level Java daemon that uses a TUN/TAP device to interpose itself as an overlay between the transport and IP layer, and uses the Socket API to invoke the kernel transport protocol implementations. Most of the core features described earlier have been implemented, including dissemination of REGISTERs, name-based routing of FINDs, content caching, and protocol support for a few applications.

The implementation is comprised of the following modules:

Router Module: The router module processes incoming REGISTERs and FINDs. It maintains a memory-based registration table, computes and disseminates registrations to peers and providers in the RH topology, and routes FINDs based on the registration table. The router module also monitors the liveness of their manually configured neighbors (customers, providers, and peers) and exchanges registrations with these neighboring RHs over reliable queues implemented on TCP.

Caching Module: The caching module offers application modules (see below) access to a local cache of data items. It caches content in the local file system and uses an embedded database to maintain, on a per-item basis, metadata (which includes an expiration time) and the location of the corresponding file. A cache lookup is a two-step process: the module first queries the database to determine if the item is in the cache. If a valid cache entry is present, the module serves the cached content using file system routines.

Application Modules: To experiment with some set of applications, we augmented the prototype with a set of application modules. In each case, we provided proxies that allowed unmodified hosts to use DONA:

- *HTTP:* Components were implemented to simulate a CDN provider. A HTTP proxy module translates incoming HTTP requests into FINDs and returns content in the form of standard HTTP responses. A CDN provider uses an HTTP registrar module to submit periodic registrations. The HTTP proxy module on the provider's RH then translates the received FINDs back to HTTP to be sent to the content provider (a user of the CDN provider, which does not implement DONA). In addition, the HTTP proxy module uses the caching module to implement simple HTTP caching.

- *Session Initiation Protocol*: A SIP proxy module interfaces with legacy SIP user agents and converts their INVITE and REGISTER messages into FINDs and REGISTERs, as described in Section 3. Agents then transmit RTP traffic directly to each other.

- *Large-file distribution:* A P2P client application was developed to distribute large files. The application splits large files into chunks and creates an index file for the chunks. The application then registers and serves the chunks and index; peers missing chunks or index try periodically to download them.

- *RSS:* An RSS proxy intercepts incoming RSS refresh requests (that arrive in the form of standard HTTP GET messages from hosts that don't implement DONA) and issues TTL'ed FINDs to serve future refresh requests directly from cache, if the response contained an RSS feed (which cannot be determined reliably solely based on the HTTP request URL). The TTL'ed FIND travels to the content source and causes the intermediate RHs to establish forwarding state. On the content-provider side, a proxy registers the feed on behalf of the RSS server and polls the feed at periodic time intervals to detect changes. If updated, the proxy retrieves the new version and propagates it over the dissemination tree to the subscribers.

The prototype consists of approximately 17,100 lines of code. The core RH functionality (routing, registration maintenance, content caching) constitutes 11,900 lines while the remainder is for the application modules.

The prototype has been deployed on PlanetLab with topologies in order of tens of nodes (RHs). The deployment was useful for debugging our code, but was not of sufficient size (nor subject to sufficient workload) to allow us to study DONA's scaling properties. We discuss scaling in the next section.

6 Feasibility

Now that we have laid out DONA's many possibilities, we must discuss its basic feasibility. We first estimate the computational requirements of a single logical RH, and then we discuss one possible way to organize a large ISP's RH infrastructure.

6.1 Requirements

We discuss requirements for a domain's single logical RH to support FINDs and REGISTERs.[5] Note that an AS's RH need only keep routing state for data that lie below or equal to it in the AS hierarchy, since the default behavior is to forward FINDs to providers. Thus, the toughest requirements will be on the Tier-1 providers, whose RHs must keep everything in their registration tables, and those are the performance requirements we focus on.

Since the estimated number of public web pages is on the order of 10^{10} as of 2005 [21], we set an initial target of 10^{11} registered items; this includes all data and services accessed over the Internet. Assuming 42 bytes per entry (40 for the name and 2 for a next-hop RH), this yields a total storage of about 4TB. (This is a simplification, since it neglects the overhead of the necessary data structures. Since we are interested in orders of magnitude only, we are ignoring this complication.)

If the average lifetime of a registration is two weeks (which we are taking as an initial estimate, with little empirical support), a Tier-1 RH must handle about 83000 registration messages per second. If each registration is roughly 1 KB, then this is approximately 680Mbps, which is small compared to the aggregate bandwidth of a Tier-1 AS (and the registrations are a small fraction of the load on any particular link). However, each of these registrations involves expensive cryptographic operations. Using fast cryptosystems such as ESIGN [30], a 3 GHz processor can create and verify 2048-bit signatures in 150 and 100 microseconds, respectively [24]. If we assume a total of 500 microseconds per registration, the total registration load could be handled by roughly 40 CPUs. However, an AS may choose to trust its peering ASes to have done the crypto-checks, and thus this load could be reduced.

To estimate the rate of incoming FINDs, we assume that an RH's load will be of the same order as the current rate of HTTP requests. We realize this ignores other application protocols, but we think HTTP is sufficient to give us an order-of-magnitude estimate. As data, we use flow logs of traffic observed over one week in November 2005 at routers in two transit networks: 4 routers belonging to Abilene [1], the academic and research network within the United States; and 4 routers from GEANT [18], a large research network in Europe which also carries some commercial traffic. Sampled flow logs from each network were used to count HTTP packets.[6]

Table 1 shows the HTTP request rate and, for context, the DNS and TCP SYN rates for each log, as well as the average over the four logs. These results are normalized to show the number of requests that would occur in a full gigabit's worth of network traffic. Thus, the HTTP request rate for a fully loaded Gbps link is on the order

[5]All the other functionality discussed in Section 4 is optional (*e.g.*, an RH can decide to not cache if it can't handle the load) and/or significantly less burdensome than the basic processing of the stream of FINDs and REGISTERs.

[6]We use the number of TCP packets to port 80 to estimate HTTP requests seen at the router, as requests are not discernible using flow logs alone, and ignore any skew due to the sampling.

	Feed	DNS req./s	HTTP req./s	TCP SYNs/s
Abilene	kscy	913	16,641	2,534
	ipls	939	14,591	2,799
	dnvr	861	18,176	2,448
	losa	827	8,851	1,934
	Avg.	885	14,565	2,429
GEANT	at1	2,072	5,237	5,619
	de1	2,292	7,191	3,913
	hu1	738	16,445	3,829
	se1	1,813	10,157	2,986
	Avg.	1,728	9,757	4,087

Table 1: Rates of DNS requests, HTTP packets and TCP SYNs scaled for a fully utilized 1 Gbps link.

of 20,000 requests per second; this density of requests is consistent with a (somewhat dated) PolyMix-4 workload used to benchmark web caches [26]. We thus assume that a fully-loaded Gbps link will generate about 20,000 FINDs per second. If FINDs are 150 bytes (including transport layer header as well as headers from below), then these FINDs consume about 24Mbps, only 2.4% of the link.

FINDs can be processed either from RAM or from disk. For an RH to hold the entire registration table in RAM would require roughly 500 PCs each with 8GB of RAM (or some smaller number of servers, each with larger memories, as in the Sun Blackbox [35]). If we assume a single CPU can process roughly 40,000 FINDs per second (similar speeds have been achieved with similar loads in [39]) and use the estimated FIND density of 20,000 FINDs per Gbps, then this ensemble of PCs could handle an aggregate load of 1Tbps coming in from peering links. 500 PCs might sound like a lot, but recall that this is to serve an entire Tier-1 ISP, and these total requirements are tiny on the scale of modern datacenters.

Alternatively, processing a FIND from disk takes roughly 2.5 msec (as [39] has again shown with similar load), so handling 20,000 requests per second requires roughly 50 disks. This requires an additional 50 disks for each incoming fully loaded 1Gbps link. However, a single PC with 8 GB of RAM can hold a routing table for 1/500'th of the entire database. Presumably the cache hit rate for this fraction of the entire corpus would be fairly high (though we don't speculate as to the exact number); this would substantially reduce the load on the disks, so many fewer than 50 disks would be needed for each Gbps of incoming traffic. Moreover, for many domains lower in the AS hierarchy, the total number of items below it in the tree would be less than 1/500th of the entire database, and thus the entire routing table could be held in RAM. This would certainly hold for all enterprise and campus networks.

Whether an AS uses the all-in-memory approach or relies on disks depends on its aggregate load and other factors. However, we note that a single fully-loaded Sun Blackbox (250 multi-core processors, 7TB RAM) [35] can easily handle the memory and processing requirements of a large ISP, so this is easily within the reach of today's technology.

6.2 Design of an RH Infrastructure

The estimates above are for a domain's single logical RH (although an AS might choose to have several logical RHs to avoid inter-continental coordination). Of course, ASes are distributed, with many incoming links spread across a wide geographic area. We now discuss, in very general terms, how a Tier-1 AS might design its RH infrastructure. We follow the model of RCP [4] in centralizing the design as much as possible, thereby reducing problems of consistency and coordination.

The RH infrastructure will be comprised of a Master RH (MRH) and a set of Cache RHs (CRHs) situated in the PoPs. The MRH receives and stores all registrations, and it disseminates them further to its peers (and providers, if this is not a Tier-1 ISP). We presume the MRH will be a cluster of PCs. Depending on the size of the AS, it can choose whether to use the all-RAM approach or use disks to serve requests.

CRHs route FINDs incoming from customer, peering and transit links. If a CRH does not have an entry in its table, it forwards the FIND to the MRH. We envision a CRH to be a commodity PC, perhaps even diskless, using RAM as its primary storage. This would allow a single PC to route, as noted before, roughly 40,000 FINDs per second (for those FINDs that had the necessary entry in memory). As noted above, a PC's individual RAM would hold a sizable fraction of the database, so the cache hit rate should be substantial. However, to increase the hit rate even further, CRHs could use cooperative caching; CRHs at a single PoP could partition the entire namespace and dedicate a PC per slice.

In terms of failures, if a CRH crashes then it merely repopulates its cache incrementally as FINDs and REGISTERs arrive. If an MRH crashes, it can resynchronize with its peering MRHs to re-establish state once again operational.

7 Related Work

Almost every piece of DONA's design has been discussed in the literature. Here we give a very quick listing of some of the most relevant pieces of related work.

DONA is built on name-based routing as is advocated in TRIAD [19], which was arguably the first to explore the benefits of diverging from the classic lookup-oriented Internet architecture. We differ from TRIAD in several ways: our use of flat, self-certifying names, the extended RH functionality described in Section 4, and our consideration of administrative access policies. IPNL argued for the benefits of name-based routing in terms of improving reachability in modern NATted networks [15]. More recently, the DTN work has favored name-based routing in challenged networks (*e.g.*, [6,12,34]). Finally, perhaps a bit surprisingly, the most widely known instance of name-based routing is HTTP [13], as its proxy-architecture is essentially a name-based routing approach.

The flat self-certifying naming used by DONA has a long history, at least in the research community, starting with HIP [28] and SFS [25]. The role that this form of naming might play in more general Internet architectures has been discussed in, *e.g.*, [2,37,38]. These proposals focused on naming and name resolution, as DONA does; however, routing directly on flat names, at line rates, has also recently gained some interest, although its scalability remains an open question [5,23]. Finally, the public-key-centric approach used by DONA has shown its value in policy-enforcement, as the SPKI efforts [9] testify.

The importance and applications of anycast have been discussed in many papers, and several in the research community have focused on building shared anycast infrastructures (*e.g.*, [16, 29]). In a similar manner, DONA's focus on data availability has been shared by recent efforts to improve availability through augmenting inter-domain routing (as in [40]) or by isolating the applications entirely from the transfer of data (as proposed in [36]). DONA builds on these results and incorporates anycast and data-orientation deeper into the network architecture.

Involving all stakeholders along the path, and incorporating middleboxes, via DONA's signaling primitives is aligned with more recent efforts in IRTF. NUTSS [20] and EME RG [10] propose using SIP as a generic signaling medium, whereas we are taking a more clean-slate approach. Inclusion of end-point imposed middleboxes has been considered in [2,38].

8 Broader Architectural Implications

Currently, applications obtain a hostname from some out-of-band mechanism, issue a DNS query to translate that hostname into an address, and then invoke the Socket API to initiate a byte-stream transfer to that host. Thus, applications are oriented around hosts, addresses, and bytes. However, with DONA, one could instead use an application interface that is based on the FIND and REGISTER primitives. This change has several obvious benefits, but it also leads us towards deeper issues, about which we can only speculate.

With such a data-oriented application interface, the applications would revolve around the names of data and services, not the address or hostname of their location. We have already discussed the advantages that accrue from this, in terms of persistence, authentication, and availability. The interface would also enable application protocols to be oriented around application objects (*i.e.*, messages), not bytes, because that is the granularity of naming. This not a new concept, as application-layer framing has been advocated in [7] and adopted by many modern protocols (such as SCTP [33]); here we are merely noting that it fits well within the DONA approach.

Perhaps most importantly, raising the level of abstraction in the application interface would shield applications from low-level communication details. At a practical level, this shielding would allow the low-level data communication protocols to be selected dynamically. Transport could be tailored to the context (as in [36]) and extremely opportunistic, using whatever application and transport protocols are available in the current operating context (as suggested in Haggle [34], for instance), including bluetooth, USB keys, and other non-IP mechanisms. In the extreme, when REGISTERing data, the application would essentially transfer custody of the data to lower levels of the protocol stack (*i.e.*, those that would respond to a FIND request) and need not be aware of any future retrieval. This would cleave the portion of the application that served content from the portion that decided to share it.

In a similar manner, the FIND and REGISTER propagation mechanisms could become context-dependent; while the mechanisms we presented here are appropriate for a well-connected Internet infrastructure, one could use other methods (such as flooding FINDs and merely caching REGISTERs locally) in *ad hoc* networks or other challenged contexts.

At a more paradigmatic level, this new data-oriented application interface is semantically similar to a publish (REGISTER) and subscribe (FIND) interface. Pub/sub effectively decouples the application end-points in space, time, and synchronization [11] and is used in many contexts. The universality of such an interface raises the possibility that a new pub/sub-based spanning layer [8] might emerge, stretching over the Internet of today, as well as over networks beyond it. We don't know whether DONA is the right way of implementing this new spanning layer, but we hope to explore the extent to which DONA can lead us down this intriguing path.

Acknowledgments

We thank the anonymous reviewers for their comments. We gratefully acknowledge stimulating conversations with David Cheriton, Van Jacobson, Pekka Nikander, Sylvia Ratnasamy and many others. DONA is a continuation of a line of joint research with Michael Walfish and Hari Balakrishnan; our dialogue is the bedrock on which this work rests, and their insightful comments helped shape and sharpen our earlier drafts. This work was supported by the NSF under grants CNS-0225660, CNS-0205519, CNS-052041, and by British Telecom.

9 References

[1] Abilene. `http://abilene.internet2.edu`.

[2] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proc. of ACM SIGCOMM '04*, pages 343–352, Portland, OR, USA, Aug. 2004.

[3] M. Blumenthal and D. Clark. Rethinking the design of the Internet: The End-to-End arguments vs. The Brave New World. *ACM TOIT*, pages 70–109, 2001.

[4] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. of NSDI '05*, pages 15–28, Boston, MA, USA, May 2005.

[5] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, S. Shenker, and I. Stoica. Routing on Flat Labels. In *Proc. of ACM SIGCOMM '06*, pages 363–374, Pisa, Italy, Sept. 2006.

[6] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay Tolerant Networking Architecture. Internet Draft, IETF, Dec. 2006.

[7] D. Clark and D. Tennenhouse. Architectural Consideration for a New Generation of Protocols. In *Proc. of ACM SIGCOMM '90*, pages 200–208, Philadelphia, USA, 1990.

[8] D. D. Clark. Interoperation, Open Interfaces, and Protocol Architecture. *The Unpredictable Certainty, Information Infrastructure Through 2000: White Papers*, pages 133–144, 1997.

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. SPKI Certificate Theory. RFC 2693, IETF, Sept. 1999.

[10] End-Middle-End Research Group. `http://www.irtf.org/charter?gtype=rg&group=eme`.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[12] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proc. of ACM SIGCOMM '03*, pages 27–34, Karlsruhe, Germany, 2003.

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

[14] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent Personal Names for Globally Connected Mobile Devices. In *Proc. of OSDI 2006*, pages 233–248, Seattle, WA, USA, Nov. 2006.

[15] P. Francis and R. Gummadi. IPNL: A NAT-extended Internet Architecture. In *Proc. of ACM SIGCOMM '01*, pages 69–80, San Diego, CA, USA, Aug. 2001.

[16] M. J. Freedman, K. Lakshminarayanan, and D. Mazières. OASIS: Anycast for Any Service. In *Proc. of NSDI '06*, pages 129–142, San Jose, CA, USA, May 2006.

[17] D. Funato, K. Yasuda, and H. Tokuda. TCP-R: TCP Mobility Support for Continuous Operation. In *Proc. of ICNP '97*, pages 229–236, Atlanta, GA, USA, 1997.

[18] GEANT2. `http://www.geant2.net`.

[19] M. Gritter and D. R. Cheriton. TRIAD: A New Next-Generation Internet Architecture. `http://www-dsg.stanford.edu/triad`, July 2000.

[20] S. Guha and P. Francis. An End-Middle-End Approach to Connection Establishment. In *Proc. of ACM SIGCOMM '07*, Kyoto, Japan, Aug. 2007.

[21] A. Gulli and A. Signorini. The Indexable Web Is More Than 11.5 Billion Pages. In *Special Interest Tracks and Posters of The 14th International Conference on World Wide Web, WWW '05*, pages 902–903, Chiba, Japan, 2005.

[22] M. Handley and A. Greenhalgh. Steps Towards a DoS-resistant Internet Architecture. In *Proc. of ACM SIGCOMM FDNA '04*, pages 49–56, Portland, OR, USA, Aug. 2004.

[23] D. V. Krioukov, K. R. Fall, and X. Yang. Compact Routing on Internet-like Graphs. In *Proc. of IEEE INFOCOM*, Hong Kong, Mar. 2004.

[24] J. Li, M. N. Krohn, D. Mazières, and D. Shasha. Secure Untrusted Data Repository (SUNDR). In *Proc. of OSDI 2004*, pages 121–136, San Francisco, CA, USA, Dec. 2004.

[25] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating Key Management from File System Security. In *Proc. of SOSP '99*, pages 124–139, Charleston, SC, USA, Dec. 1999.

[26] Measurement Factory. Public Benchmarking Results. `http://www.measurement-factory.com/results`, Dec. 2001.

[27] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. Internet Draft, IETF, Dec. 2006.

[28] R. Moskowitz and P. Nikander. Host Identity Protocol Architecture. RFC 4423, IETF, May 2006.

[29] A. Nandi, A. Ganjam, P. Druschel, T. Eugene, I. Stoica, H. Zhang, and B. Bhattacharjee. SAAR: A Shared Control Plane for Overlay Multicast. In *Proc. of NSDI '07*, pages 57–72, Cambridge, MA, USA, Apr. 2007.

[30] T. Okamoto and J. Stern. Almost Uniform Density of Power Residues and the Provable Security of ESIGN. In *ASIACRYPT*, volume 2894 of *LNCS*, pages 287–301, Dec. 2003.

[31] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, IETF, June 2002.

[32] M. A. C. Snoeren. *A Session-based Architecture for Internet Mobility*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, Feb. 2003.

[33] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960, IETF, Oct. 2000.

[34] J. Su, J. Scott, P. Hui, E. Upton, M. H. Lim, C. Diot, J. Crowcroft, A. Goel, and E. de Lara. Haggle: Clean-slate Networking for Mobile Devices. Technical Report UCAM-CL-TR-680, University of Cambridge, Computer Laboratory, Jan. 2007.

[35] Sun Microsystems. Project Blackbox. `http://www.sun.com/blackbox/`, Jan. 2007.

[36] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An Architecture for Internet Data Transfer. In *Proc. of NSDI '06*, pages 253–266, San Jose, CA, USA, May 2006.

[37] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Proc. of NSDI '04*, pages 225–238, San Francisco, CA, USA, Mar. 2004.

[38] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *Proc. of OSDI 2004*, pages 215–230, San Francisco, CA, USA, Dec. 2004.

[39] M. Walfish, J. D. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed Quota Enforcement for Spam Control. In *Proc. of NSDI '06*, pages 281–296, San Jose, CA, USA, May 2006.

[40] D. Wendlandt, I. Avramopoulos, D. G. Andersen, and J. Rexford. Don't Secure Routing Protocols, Secure Data Delivery. In *Proc. of Hot Topics in Networks*, Irvine, CA, USA, Nov. 2006.

[41] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proc. of IEEE Symposium on Security and Privacy*, pages 130–143, Oakland, CA, USA, 2004.

[42] X. Yang, D. Clark, and A. Berger. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking (to appear)*, Dec. 2007.

[43] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proc. of ACM SIGCOMM '05*, pages 241–252, 2005.