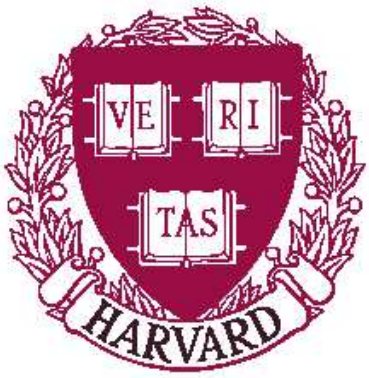
A Directory Service for Perspective Access Networks

Geoffrey Goodell
Mema Roussopoulos
and
Scott Bradner

TR-06-06



Computer Science Group
Harvard University
Cambridge, Massachusetts

A Directory Service for Perspective Access Networks

Geoffrey Goodell Mema Roussopoulos Scott Bradner

Abstract

Network fragmentation occurs when the accessibility of a network-based resource to an observer is a function of how the observer is connected to the network. In the context of the Internet, network fragmentation is well-known and occurs in many situations, including an increasing preponderance of network address translation, firewalls, and virtual private networks. Recently, however, new threats to Internet consistency have received media attention. Alternative namespaces have emerged as the result of formal objections to the process by which Internet names and addresses are provisioned. In addition, various governments and service providers around the world have deployed network technology that (accidentally or intentionally) restricts access to certain Internet content. Combined with the aforementioned sources of fragmentation, these new concerns provide ample motivation for a network that allows users the ability to specify not only the network location of Internet resources they want to view but also the *perspectives* from which they want to view them. Our vision of a *Perspective Access Network* is a peer-to-peer overlay network that incorporates routing and directory services that allow non-hierarchical organization. In this paper, we present the design, implementation, and evaluation of a directory service for such networks. We demonstrate its feasibility and efficacy using measurements from a test deployment using PlanetLab.

I. Introduction

Network fragmentation occurs when the availability of a resource to an observer is a function of how the observer is connected to the network. In the context of the Internet, network fragmentation is well-known and occurs in many situations, including an increasing preponderance of network address translation, firewalls, and virtual private networks.

Recently, however, new threats to Internet consistency have received media attention. First, a number of nations have raised formal objections to US oversight of ICANN, the organization responsible for provisioning Internet names and addresses [1], and a number of private organizations such as UnifiedRoot have emerged to offer alternative namespaces [22]. Global agreement on Internet governance is becoming increasingly difficult [33] which means the potential for inconsistency in naming resulting from multiple DNS roots or addresses that are not globally unique will only increase. Second, a perceived increase in online criminal activity has created viable business models for businesses that provide geolocation services marketed for their benefits in fraud resolution and digital rights management [2]. For example, a number of companies use these geolocation services to obtain information about how a user is connected to the Internet (such as IP address and ISP data) to determine whether the user is likely to be fraudulent. This has caused a number of legitimate online transactions to be denied when users are not connected at their usual point of attachment [15]. Finally, various governments and service providers around the world have deployed network technology that (accidentally or intentionally) restricts access to certain Internet content [20], [12].

Combined with the aforementioned sources of fragmentation, these new concerns provide ample motivation for a network that would allow users the ability to specify not only the network location of Internet resources they want to view but also the *perspectives* from which they want to view them. In this paper, we present the design, implementation, and evaluation of a *Perspective Access Network*, an overlay infrastructure for sharing perspectives. Our prototype, called PAN consists of an unstructured, peer-to-peer overlay of *forwarders* carrying TCP traffic that act as intermediaries between nodes that cannot communicate directly.

Previous work on overcoming network fragmentation to facilitate end-to-end connectivity requires extensive changes to operating systems (such as deployment of new protocol stacks), requires the explicit participation of ISPs and content providers, or imposes a global hierarchical organization of the Internet. We relax these constraints to provide *ease of deployment* and have built a system we have deployed on the Tor anonymity network [8] and on PlanetLab [13]. Our approach does not require changes to the operating system or protocol stack, does not require active participation of ISPs, and does not require special configuration of in-band network-layer elements such as routers or middleboxes.

PAN also does not impose global hierarchical organization of the Internet. Currently, both the addresses and the names used to identify resources on the Internet are allocated by a collection of governance organizations that are arranged hierarchically with a single organization at the top having overall "control." Our approach allows for an Internet without hierarchically ordained names and address spaces—that is, an Internet consisting of (possibly overlapping) network fragments, each with its own local naming and addressing scheme. This scheme promotes *locality in naming*, in that

[1] Internet Corporation for Assigned Names and Numbers, `http://www.icann.org/`

[2] CyberSource, `http://www.cybersource.com/`; NatGeo `http://www.natgeo.com/`; Quova, `http://www.quova.com/`

multiple resources with the same name can co-exist in different local namespaces (fragments). This scheme also promotes *distributed management* of local networks, in that adding a new local network and its abundance of resources to the Internet need not require specific allocation of names, addresses, or routing from centralized authorities.

For our overlay, we assume that each forwarder need only have the ability to communicate bidirectionally with some subset of the other forwarders. A PAN client that wishes to view a resource from the perspective of a particular forwarder F uses the PAN distributed directory service (provided, in our prototype, by a subset of the forwarders) to determine a path of connectivity through the forwarders to F. The PAN client then constructs a source-routed circuit through the forwarders on the path to F, which then performs a DNS lookup to resolve the local resource name to an IP address from its point of view and accesses the resource on behalf of the client. The client therefore accesses the resource from the perspective of F.

Since we do not impose a global unique naming scheme for resources, we need a way to uniquely identify a resource. We therefore require forwarders to generate unique, self-certifying identifiers and a PAN client specifies a particular resource by concatenating the forwarder ID with the resource name as resolved the forwarder. This design choice, however, sacrifices a certain amount of aggregation we can perform when advertising forwarder route information within the PAN overlay.

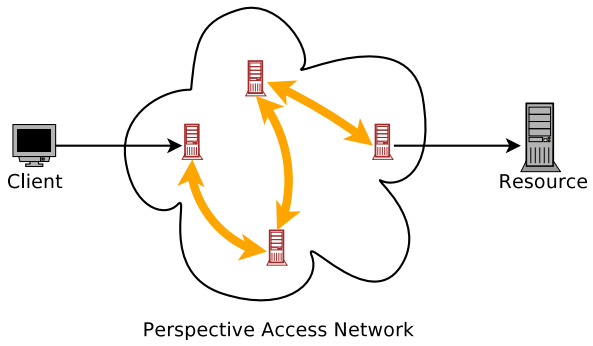


Fig. 1. PERSPECTIVE ACCESS NETWORK OVERVIEW. PAN presents a peer-to-peer network for sharing perspectives, allowing access to resources in circumstances in which the meaning of names and addresses is a function of their context.

This paper focuses on the directory service that enables perspective-sharing in PAN. After comparison with previous work (Section II), we describe the design and implementation of our directory service (Section III). We provide an analysis and evaluation of the service through measurements from a deployment of PAN nodes using PlanetLab. In particular, we explore the tradeoffs that arise as a result of our non-hierarchical design choice, particularly the fact that the extent to which we can take advantage of aggregation is limited (Section IV). Finally we conclude with a brief discussion of future work considering policy issues involved in the deployment of PAN (Section V).

II. RELATED WORK

A number of existing projects that focus on overcoming Internet fragmentation propose their own directory management schemes. These projects include:

- INDIRECTION. I3 [26] provides a "rendezvous-based communication abstraction" in which providers of services register with a particular location in the network, and those peers requesting services communicate with that location rather than with the provider directly. TRIAD [2] uses globally unique, hierarchical names to identify networks; these names are propagated throughout the system via BGP-like advertisements among TRIAD nodes. PAN does not require registration of services, names of resources need not be globally unique, and names of PAN forwarders are non-hierarchical.
- ANTI-CENSORSHIP. Psiphon is a single proxy application used to circumvent content filtering. A host within a country without filtering installs the Psiphon proxy software and remote hosts in countries with filtering can access blocked web sites through the proxy. Infranet [10] and Tor [8] use overlay networks to provide anonymous communication. Anonymity networks such as these can also be used for anti-censorship purposes, specifically to circumvent local restrictions on access to resources. However, since the Internet is not entirely flat, the resources to which a user of these networks (or of Psiphon) has access may vary as a function of the particular overlay node (or Psiphon host) that is used as the last-hop proxy. For example, requesting a particular web page from an anonymity network might yield content that has been tailored to the particular local network or geographic region in which the last-hop proxy resides. If anonymity is the goal, then a larger anonymity set may be worth the cost of some probabilistic variation in content reachability. PAN takes the opposite approach, choosing to use an overlay proxy network to maximize content reachability, possibly at the expense of anonymity.
- DECOUPLING POLICY FROM MECHANISM. FARA [3], [4] provides a general framework for describing associations between nodes without requiring a global namespace. Platypus [24] provides a system for enforcing routing policy on the forwarding plane rather than the control plane, relying upon cooperation from intermediary ISPs. PAN aims not to require such cooperation, at least not on a technical level. However, PAN does present an argument for separating network access policy from technical decisions made at the network layer. If two PAN forwarders are both connected to the same PAN overlay, then technically speaking, each could have access to whatever the other can see, regardless of what lies between.
- NON-UNIVERSAL NAMESPACES. Semantic-Free Referencing [31] stipulates that resources have globally-unique "semantic-free tags", high-entropy bit strings perhaps generated as self-certifying names by the resource provider. A client would use the semantic-free tag rather than a hostname to identify the website, and a Reference

Resolution Service (RRS) would map human-readable names to semantic-free tags. The goal is to decouple the name of a resource from its content; note that this is subtly different from the *naming locality* goal of PAN. The possibility of having multiple different RRS servers suggests that this approach could lead to a form of locality, since different local regions or classes of organizations could use different RRS servers to canonicalize human-readable names. The authors provide little discussion of how multiple RRS servers could conceivably exist in practice, or why a single RRS infrastructure similar to DNS would not emerge, other than to suggest that there could be a competitive market.

- EMBRACING HETEROGENEITY. Plutarch [7] takes the leap of considering network fragmentation as the inevitable result of political or economic forces rather than some technical obstacle to be overcome. The authors convincingly argue that avoiding global management would promote innovation. Like PAN, Plutarch does not require a well-defined Internet core or global names. Plutarch "contexts" are similar to the "fragments" that we describe. However, like IPNL and unlike PAN, Plutarch requires these contexts to be well-defined and non-overlapping. Moreover, Plutarch requires special configuration of middleboxes that serve as the boundaries between contexts. Plutarch also resolves names via a peer-to-peer search, which PAN avoids in favor of reducing overhead and improving connection setup time.

The functionality provided by a directory service is necessary in a wide variey of distributed systems and networks. We cannot do justice by describing all directory services that have been proposed in the literature, so we focus on systems that are widely in use today:

- DNS. The Domain Name Service [17], [18] is the widely used directory service for resolution of hostnames and IP addresses in the Internet. DNS names are constructed and resolved, and updates are propagated across DNS servers in a hierarchical manner. The PAN forwarder ID space is flat because forwarders use self-generated, self-certifying identifiers. This means PAN directory servers can neither take advantage of the hierarchical approach of DNS nor can perform aggregation of forwarder identifiers as they propagate forwarder information through the directory service. The latter approach is that used by BGP [25], which aggregates prefix information to reduce the number of entries BGP has to carry and store. We explore the design tradeoffs that arise from our approach in Section IV.
- FILESHARING NETWORKS. Peer-to-peer file sharing systems dominate Internet traffic today. These systems require functionality that allows peers to resolve files (or file attributes) of interest to IP addresses of hosts that store the files. Some peer-to-peer systems use a centralized approach to providing this lookup functionality. For example, Napster placed the entire index of (filename, IP address) mappings on a single host. Apart from the potential scalability concerns, this approach assumes clients can access the centralized index. In PAN, we build our directory service taking into account that the Internet is fragmented and not all clients can necessarily reach one single directory server. Distributed Hash Tables (DHTs) (such as CAN [21] and Chord [27] distribute this load across the participating peers. DHTs tightly control both the placement of mapping on peers and the overlay topology which allows the efficient lookup across the overlay from a querying peer to a peer with the mapping. DHTs also assume that peers will be able to bidirectionally communicate with the peers that have been assigned to be their neighbors barring transient network partitions. Finally, '*unstructured* peer-to-peer file sharing networks, such as Gnutella [3] provide an "ad hoc" directory lookup service in that lookup queries flood the network in search of a peer who may have the mapping of interest. PAN is designed with the goal of minimizing connection setup latency for clients connecting to arbitrary services. Thus, clients do not request forwarder information via flooding because connection set up latency would grow quickly with population size. In contrast, file-sharing networks, miminizing the lookup time is not of priority because file download time dominates lookup time.
- COOPERATIVE WEB CACHING. Various systems been proposed to allow groups of participating caches to track what web objects are cached at what proxies and to exchange cached web content amongst themselves. The overall goal is to bring a particular web object to the cache that is closest to the clients requesting that web object. Previous proposals include hierarchical cache schemes (e.g., [1], [14], [32], [5]), hash-based schemes [14], [30], directory-based schemes [9], [16], [28], and multicast-based schemes (e.g., [29]). All of these schemes assume that any proxy participating in the a cooperative caching scheme can communicate bidirectionally with any other proxy.

III. ARCHITECTURE

PAN consists of a pairwise-connected overlay network of *forwarders*, each of which has access to some set of Internet resources. Some resources may be available to some nodes but not others. The overlay network that connects all of the forwarders to each other includes a *data plane* that carries tunnelled DNS requests and TCP sessions, as well as a *control plane* that carries routing information.

There are a number of problems with a distributed approach to assigning names in a network. For example, two network components may find themselves with the same name, and there are performance costs associated with choosing names that do not inherently carry location information. However, for the purposes of PAN, it is both possible and beneficial to sacrifice global agreement abount names without undermining network integrity and functionality.

To address the concern about uniqueness of names used to identify forwarders, we allow each forwarder to generate

[3]Gnutella Protocol Specification, `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`

a self-certifying identity (such identities may be mapped to human-readable nicknames by third-party certification authorities). Each forwarder, then, possesses two names: a *global* name, used to identify itself within the PAN network, and a *local* name, used to identify itself within its local namespace. By considering that each forwarder provides access to resources within its own local namespace, we avoid requiring that all names for all Internet resources be globally unique.

To specifically identify each Internet resource, we concatenate the locally meaningful name of the resource with an identifier specifying the name of the forwarder from which we want to access that resource. For the purpose of our implementation, we assume that resources are named by hostname or IP address, so to access a resource listening on TCP port 80 of 192.168.0.3 as seen by a forwarder named serifos, we would use 192.168.0.3.serifos.exit:80.

A. PAN Directories

A subset of PAN forwarders also serve as directory servers, so every PAN directory server is also a forwarder. Each directory server provides a set of *records*: (a) a *master record*, containing attributes describing itself, (b) a set of *directory records*, each containing attributes describing directory peers, and (c) a set of *forwarder records*, each containing attributes describing individual PAN forwarders. The records are published via a simple server that responds to queries in the form of HTTP-GET requests, and these attributes are periodically pushed to neighboring directories via directory updates in the form of HTTP-POST requests. Figure 2 illustrates one possible set of records stored in a directory server given one possible network of directory servers and standalone forwarders.

1) Master Records: A complete PAN directory server listing includes exactly one *master record*, which contains three attributes, as follows: a *header* consisting of the name of the directory server and its version, a *timestamp* indicating when this directory listing was created, and a *status* record identifying each forwarder indexed by the directory including a bit that indicates whether the directory believes that forwarder to be active. The bit specifying whether a given forwarder is reachable is set to true when the directory server receives a sufficiently recent descriptor for an individual forwarder, and it is set to false when the descriptor expires.

2) Directory Records: Each PAN directory server publishes a number of *directory records*, each containing a set of attributes that describe a specific peer directory server. A directory server accrues a set of directory records over time via directory updates from its neighbors. Unlike peer-to-peer filesharing services such as Gnutella or BitTorrent, PAN is designed with the goal of minimizing connection setup latency for clients connecting to arbitrary services. Thus, clients do not request forwarder records via broadcasting or heuristic searches; instead, each directory maintains a set of directory records, each uniquely corresponding to one of its peers. Scalability dictates that each individual directory server need not know everything about the entire network, so there is no guarantee that each directory server contains a record for each other directory server in the entire network.

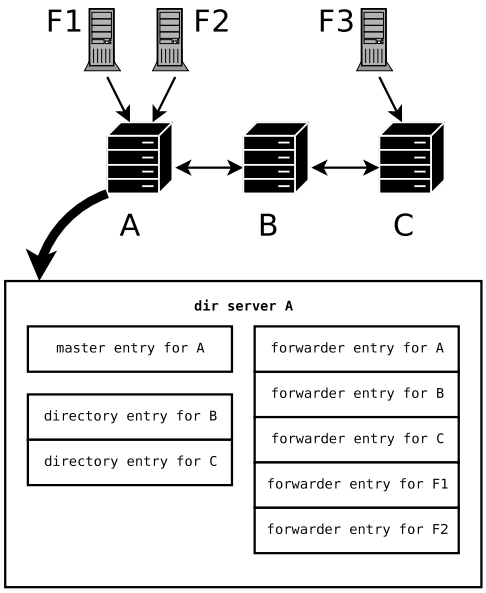


Fig. 2. RECORDS IN PAN DIRECTORIES. Given three directory servers $\{A, B, C\}$ and two standalone forwarders $\{F_1, F_2\}$ as shown at right, the table at left illustrates one possible set of records published by directory server A.

When a client issues a query for a forwarder record, but a directory server has no corresponding forwarder record, the directory server may refer the client to a set of directory servers that have previously indicated knowledge of forwarder records matching the request of the client. This *referral* consists of a set of directory records and the forwarder records that correspond to the directory servers.

Since directories are not required to explicitly fetch information on behalf of their clients, a client that queries a directory for information can expect to be referred to a specific neighboring directory server. However, such referrals are not arbitrary: clients seeking a particular forwarder record will be sequentially referred to some subset of the set of directories along the reversal of the path by which the advertisement of the forwarder propagated through the network.

We use ABNF [6] to specify the format of text fields. We specify self-certifying forwarder names and metadata fields according to the following formats:

```
FNAME := 40(ALPHA / DIGIT)

FMETA := *(ALPHA / DIGIT / "-")
```

Each directory record contains the following attributes:

- SERVICE-DESIGNATION. *(required)* This field tells a client how to connect to a directory server, given that the client has already constructed a circuit to the forwarder residing on the same machine as the directory server. In our present implementation, this field is a TCP port number. *Format:* *VCHAR
- PROPAGATION-PATH. *(required)* This field contains an ordered list of directory servers through which this particular directory record has propagated before reaching the directory server upon which it presently resides. The primary purpose of this field is to avoid cycles in

the propagation of directory records. The value of this attribute may be empty, in which case the propagation path for this particular directory record is presumed to be the empty list (i.e., the directory server described by this record is a neighbor of the directory server upon which this record presently resides). *Format:* `*1FNAME *("," FNAME)`

- SUMMARY. *(optional)* This field provides a list of PAN forwarders associated with this particular directory record, indicating that the corresponding directory offers to forward traffic to the indicated set of PAN forwarders. For each forwarder in the list, this attribute also includes a numeric value indicating the stated number of hops in the forwarding path leading to that forwarder. Note that descriptors for the forwarders indicated in this list may or may not be published at the particular directory server. See section III-B.1 for details. *Format:* `FNAME "=" *DIGIT *("," FNAME "=" *DIGIT)`
- COMPILED-METADATA. *(optional)* Propagation of metadata is analogous to propagation of individual forwarder descriptors. Just as *summary* attribute provides a list of forwarder names whose descriptors that might be found by querying some particular directory server, this field provides a compiled list of metadata strings that might be found by querying the specified directory server. In general, this field is a list of metadata strings representing the union of all of the metadata strings corresponding to all of the forwarders that appear in the *Summary* field of this directory record. Therefore, directory servers **may** issue referrals to clients querying for forwarder records matching some particular metadata field in the same manner by which they **may** issue referrals to clients querying for specific forwarders by name. *Format:* `FMETA *("," *FMETA)`

3) Forwarder Records: When a PAN forwarder publishes its descriptor, metadata, and connection information to some directory server, the directory server in turn creates a forwarder record using that information. Each forwarder listed in a directory has exactly one corresponding forwarder record. In general, forwarder records are updated more frequently and propagated less widely than directory records; see Section III-C for details. A directory server **must** publish a forwarder record for itself. Each forwarder record contains some subset of the following fields:

- FORWARDER DESCRIPTOR. *(required)* PAN directory servers provide *descriptors* that can be used by the PAN client to establish circuits through the forwarding network. Descriptors are self-signed statements published by forwarders that contain contact information, including IP address and port for accepting circuit-building connections, public key, and salient information about the capabilities of the forwarder, including exit policy and bandwidth measurements.
- PROPAGATION-PATH. *(required)* This field contains an ordered list of directory servers through which this particular forwarder record has propagated before reaching the directory server upon which it presently resides. The primary purpose of this field is to avoid cycles in the propagation of forwarder records. The value of this attribute may be empty, in which case the propagation path for this particular forwarder record is presumed to be the empty list (i.e., the forwarder described by this record published its information directly to the directory server upon which this record presently resides). Note that this path is not necessarily the same as that provided by the *Forwarding-Path* attribute. *Format:* `*1FNAME *("," FNAME)`
- FORWARDING-PATH. *(required)* This field contains an ordered list of directory servers indicating the circuit that a client should construct to reach the forwarder described by this record. Differences between this list and the list provided by *Propagation-Path* attribute arise in two ways. First, directory servers through which a forwarder record propagates are not required to add their names to the forwarding path. Second, the PAN architecture allows forwarders to publish their descriptors in directories in locations from which those forwarders are not directly accessible; to address this, the forwarder may provide instructions by which clients can reach it from the perspective of the directory to which it publishes its information. These instructions appear in the form of a *path*, listing a particular sequence of nodes to which to connect to establish a circuit including the target forwarder; see Section III-C.2 for details. *Format:* `*1FNAME *("," FNAME)`
- METADATA. *(optional)* This attribute provides additional information (e.g. geographic region, network name, connectivity information, access to particular resources, etc.) describing the forwarder. Since it is not part of the descriptor (we presume that descriptors have their own metadata fields), it is not signed by the forwarder with its private key, and thus it may be modified at the discretion of the directory servers through which it propagates. *Format:* `FMETA *("," *FMETA)`

B. Client Interaction

Our implementation of PAN leverages the circuit-building module of Tor [8] to instruct a running Tor process to build a circuit through the overlay of PAN forwarders. To see how the various components interact, refer to Figure 3. The main PAN client process itself does not interact with client applications directly; instead, it communicates with PAN directory servers using specially-built Tor circuits, and it uses descriptors obtained from these conversations to instruct Tor to build circuits that client applications can use. To take advantage of PAN, client applications may need to interact with an application-specific proxy that assures that requests for network resources are semantically correct. For example, a proxy for a web browser might rewrite HTTP headers to excise the PAN forwarder request from the hostname fields. Similarly, the same proxy might rewrite HTML tags containing URLs to ensure that all links on a page are accessed via the same PAN directives when clicked or loaded automatically.

1) Issuing Queries: To establish a path to a specified exit point, PAN must first determine the path to the exit point and

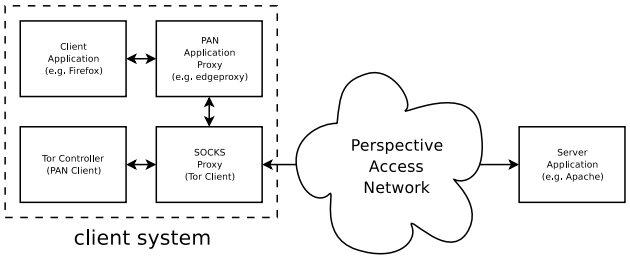


Fig. 3. CLIENT PERSPECTIVE. Client applications communicate with PAN via a series of proxies; PAN consists of software (a program that controls a running Tor process) as well as a service (the perspective access network itself).

obtain descriptors for each of the forwarders along that path, including the last one. Sufficient information necessary to learn a path to a given destination and all of the requisite descriptors may be available from the directory server to which the client speaks directly. Otherwise, the client will need to obtain the missing information via a series of queries to directory servers. See Figure 4. Each time that a client queries a directory server *A* and is referred to another directory server *B* for more information, the client extends the circuit used to communicate with *A* to *B*, thus adding a single hop to the circuit.

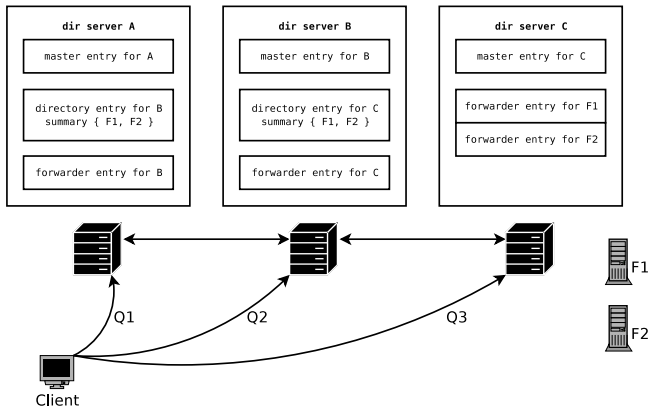


Fig. 4. ISSUING QUERIES. Suppose that a client application requests a service as seen by forwarder F_2, and the PAN client is configured to use directory server *A*. The client first sends a query to *A*, who responds with a referral to *B*. The client next sends a query to *B*, who in turn refers it to *C*. Finally the client sends a query to *C*, who has the descriptor. The client then uses the resulting circuit through $\{A, B, C\}$ to connect to the target service.

There are two types of queries, *specific queries* and *general queries*. *Specific queries* request a path to a particular forwarder whose name matches a given string, indicating that the client wants to build a circuit that terminates at some specific last-hop forwarder. *General queries* request a path to a forwarder whose *Metadata* field matches some particular string, indicating that the client wants to build a circuit that terminates at any last-hop forwarder whose forwarder record on some directory server matches some criterion. Note that directories control the content of *Metadata* fields within forwarder records, so, for example, a client issuing a general query **may** choose to reject a circuit to a specific forwarder if its descriptor does not contain a metadata record matching the original request.

The contract between a directory server and a client issuing a query is as follows. If a client issues a query, then the directory server **must** respond with one of the following:

- (a) a forwarder record for a forwarder that matches the query,
- (b) (in the event of a specific query) some set of directory records and their corresponding forwarder records, such that each directory record contains either a *Summary* field containing an element that matches a given forwarder name,
- (c) (in the event of a general query) some set of directory records and their corresponding forwarder records, such that each directory record contains a *Compiled-Metadata* field containing an element that matches a given string, or
- (d) an empty list of records, indicating that the query was unsuccessful.

Finally, a directory server **may** interpret a query as *recursive*, meaning just as some DNS servers are configured to issue DNS requests on behalf of their clients, PAN directory servers may issue queries on behalf of their clients, provided that they return results that satisfy the criteria listed above. One incentive to configure directory servers to perform recursive queries is that it reduces the amount of work and network activity on the part of the client.

A client may specify to the directory server that it intends for its query to be non-recursive, in which case the directory **should** honor that request.

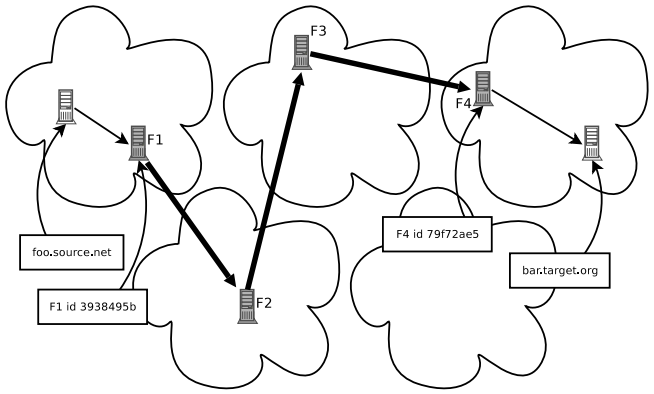


Fig. 5. ACCESSING A RESOURCE. After making use of the PAN directory servers, a client system has a source route suitable for building a circuit through the set of forwarders to the last-hop forwarder, through which the client can access the (otherwise occluded) Internet resource.

2) Building Circuits: In our prototype, once it has obtained forwarder records for the entire path to the last-hop forwarder, the PAN client will provide the necessary descriptors to Tor and then ask Tor to build a circuit using those descriptors (see Figure 5). Once the circuit has been built, PAN will inform Tor that the TCP stream received from the client application should be attached to the newly constructed circuit. We have used our implementation[4] to confirm that the set of web pages

[4]Blossom, `http://afs.eecs.harvard.edu/~goodell/blossom/`

accessible from some ISP in China differs from the set of web pages accessible from some ISP in Boston.

C. Directory Protocol

The directory servers propagate both forwarder records and directory records to other directory servers throughout the system. In this manner, any client using any of the directory servers throughout the system will have a measure of assurance that it can build a circuit to its requested forwarder, provided that directory server configuration permitted the propagation of routing information.

Directory records are stored as *long-term state* that is assumed to be up-to-date unless a *Directory Update* request from a neigboring directory server is received. Since the message volume involved in maintaining synchronicity of routing information can be expensive, only the changes are pushed from a directory to its neighbors. When a directory first comes online, and periodically over a long time interval thereafter, it requests a *burst* from each of its neighbors. The burst contains all of the directory records that the neighbor would ordinarily provide via regular directory updates, reflecting the state of knowledge that the requestor would have had if it had been receiving directory updates since the neighbor first came online. After receiving the bursts, the requestor applies a path-selection algorithm to determine the set of records that it should propagate, and it updates each of its neighbors with this set of records. Subsequently, the directory will only receive *directory updates* from its neighbors when individual records change. Each time the directory server receives a directory update that results in a change to its own set of records, that directory server **should** notify its neighbors about the change within a reasonable period of time.

Conversely, forwarder records are stored as *short-term state* that is periodically refreshed, since forwarder descriptors change frequently and individual forwarders themselves may join and leave the network frequently. Individual forwarder records must be periodically re-issued: if a forwarder record becomes too old before it is replaced, then directory servers **should** discard it.

Periodically, neighbors send empty updates to each other, even if they have no directory changes to send. Such empty updates are *keepalive* messages. If a directory has not heard from one of its neighbors for a sufficiently long period of time, it concludes that the link to the neighbor has been severed and responds by issuing a *withdrawal* message to its peers indicating that the directory record is no longer available. Withdrawal messages carry valid *Propagation-Path* attributes, and any directory server that currently offers a directory record whose *Propagation-Path* attribute contains the name of a neighbor from which it received a withdrawal message **must** either propagate to its neighbors either the withdrawal message itself or an ordinary directory record with a *Propagation-Path* attribute that does not contain the name of the neighbor from which it received the withdrawal.

1) Directory Propagation: Both directory records and forwarder records are propagated using a BGP-like path-vector protocol that includes a simple route selection algorithm

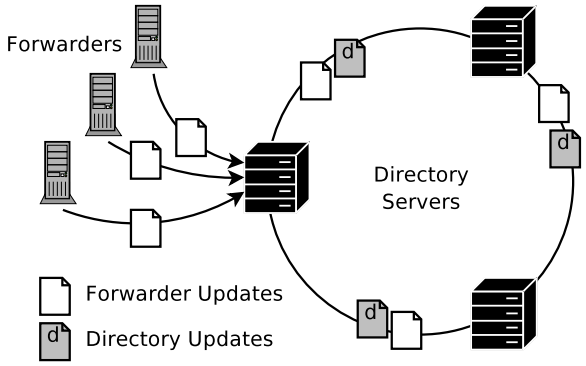


Fig. 6. DIRECTORY PROPAGATION. Each forwarder publishes its forwarder record to some set of directory servers, and each directory server publishes its directory record to its neighbors. Directory servers propagate both kinds of records according to their individually-configured policies.

applied at each directory server. Figure 6 illustrates the process by which route information is propagated through the network. Each forwarder advertises its forwarder record to some set of directory servers, and directory servers propagate the forwarder record through the network as far as policy permits. Forwarders that are also directory servers advertise only to themselves. Each directory server creates a directory record for each of its neighbor directory servers and propagates the record through the network. Thus, forwarders push forwarder records to directory servers, and directory servers push both forwarder records and directory records to other directory servers.

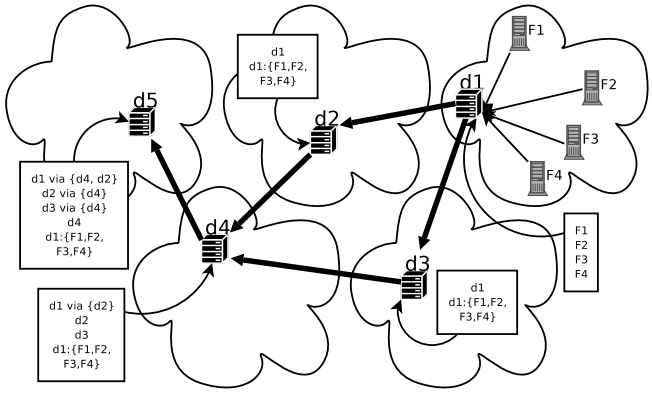


Fig. 7. ADVERTISING PAN FORWARDERS. PAN directory servers use a *path-vector* algorithm to propagate contact information for PAN forwarders. Black lines indicate the path taken by an advertisement initiated by the directory server labeled d_1. The boxes represent the records stored at the various directory servers, including *Propagation-Path* and *Summary* attributes of directory records.

If a directory server receives two conflicting forwarder records (e.g., two records with different attributes for the same forwarder), it chooses the one to propagate based upon the length of its *Forwarding-Path* attribute. If a directory server receives two conflicting directory records, it chooses the one to propagate based upon the length of its *Propagation-Path* attribute. Figure 7 provides an overview of how forwarder information propagates in the general case. The specific con-

figuration of individual directory servers may cause exceptions to these rules; Section III-D discusses this in greater depth.

2) Directory Requests: Directories address five different kinds of requests, all issued using HTTP/1.1 [11]:

- COMPLETE LISTING. This is a request for the entire set of records, including its master record, all directory records, and all forwarder records. The response to this request can potentially be quite large, but query overhead for a client could be reduced substantially if most of the forwarders to which it desires to build circuits have forwarder records published on the same directory server. *Request Format:*

  `"GET /pan/ HTTP/1.1"`

- DIRECTORY BURST. This is a special request sent by a directory server when it first comes online to bootstrap its knowledge of the records advertised by each of its neighbors. A directory server responds to this request by providing a master record, all of its hard state (i.e. all directory records), and its own forwarder record. *Request Format:*

  `"GET /pan/burst HTTP/1.1"`

- QUERY. This is a query from a client or directory server for a forwarder record, either explicitly (by name) or implicitly (by metadata or descriptor-derived data field). See Section III-B.1 for details. *Request Format:*

  `"GET /desc/id/" FNAME SP "HTTP/1.1"`

  `"GET /desc/meta/" FMETA SP "HTTP/1.1"`

- PUBLISH FORWARDER RECORD. This is a request from a forwarder to store a complete forwarder record (possibly including an explicit forwarding path and metadata). *Request Format:*

  `"POST /pan/ HTTP/1.1"`

- DIRECTORY UPDATE. This is a request from a neighboring directory server to record any updates reflecting any changes to the directory of that neighbor that occurred during the last update interval. *Request Format:*

  `"POST /pan/directory-update HTTP/1.1"`

D. Directory Configuration

A number of parameters govern how individual PAN forwarders interact with forwarders, clients, and their peers. These parameters include `neighbor` directives, which specify the set of peers with whom a directory server communicates directly, `ingress` directives, which specify preferences for advertisements received from forwarders and neighbors, and `egress` directives, which specify filters for sharing routes with clients and peers. In this section, we describe the syntax and practical significance of these parameters.

1) Peering Arrangements: Directories establish peering relationships with each other in a manner similar to how autonomous systems establish peering relationships with each other in a BGP context. A special configuration file contains a list of neighbors along with peering policy and reachability information in the following format:

`"neighbor" SP FNAME SP POLICY SP HOST ":" PORT`

The `POLICY` field represents a peering directive that takes one of five values (see Figure 8 for an illustration):

- FULL. The directory server propagates both directory records and forwarder records received from the specified neighbor, adjusting the *Propagation-Path* attribute of each record by appending the name of the neighbor. In general, other fields **must** remain unmodified, though the directory server **may** alter *Metadata* and *Compiled-Metadata* attributes.
- PREPEND. The directory server propagates both directory records and forwarder records received from the specified neighbor, adjusting the propagation path by appending the name of the neighbor **and also** adjusting the *Forwarding-Path* of each forwarder record by appending its own name. Thus, clients will be instructed to build a circuit through our node en route to the destination forwarders propagated via this neighbor. Modification of other fields is subject to the same conditions that apply to the *Full* directive.
- SUMMARIZE. The directory server propagates directory records received from the specified neighbor, adjusting the propagation path by appending the name of the neighbor. However, rather than propagating all forwarder records from this neighbor, the directory server propa-

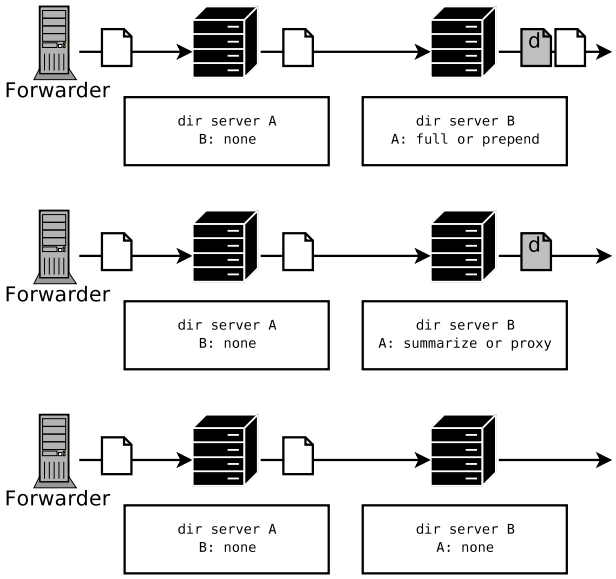


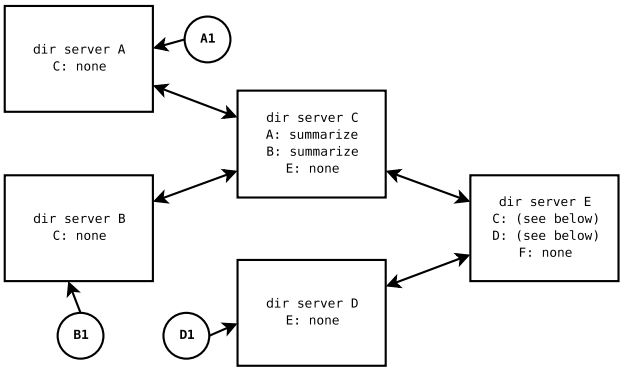
Fig. 8. PEERING DIRECTIVES. Suppose that a forwarder publishes to directory server *A*, and directory server *B* accepts updates from directory server *A* subject to some particular peering directive. If the peering directive is FULL or PREPEND, then *B* will propagate the forwarder record in addition to a directory record for *A*. If the peering directive is SUMMARIZE or PROXY, then *B* will include the name of the forwarder in the *Summary* attribute in the directory record for *A*. If the peering directive is NONE, then *B* will propagate no information about *A* or the forwarder records propagated from *A*. White pages are forwarder records; gray pages labelled `d` are directory updates.

gates only forwarder records corresponding to directory servers. In addition, the directory server creates a *Summary* attribute for this neighbor and adds the names of each forwarder whose forwarder record is received from this neighbor other than the neighbor itself. The directory server **should** define the numeric value associated with each forwarder in the *Summary* attribute as one plus the length of the *Forwarding-Path* attribute from its forwarder record. Similarly, the directory server creates a *Compiled-Metadata* attribute for this neighbor. The directory server **may** define this attribute as the union of all *Metadata* attributes included in all forwarder records received via this neighbor except the forwarder record for the neighbor itself. For each of the new attributes, if it is non-empty, then the directory server adds it to the directory record for this neighbor.

- PROXY. The directory server propagates neither directory records nor forwarder records received from the neighbor. Instead, the directory server creates a new directory record for this neighbor, according to the following specification. The *Summary* field of the new directory record **must** contain the union of all of the names of all of the forwarders from which the directory server received forwarder records from this neighbor **and** all of the names of all of the forwarders appearing in all *Summary* fields included in all directory records received from the specified neighbor. The directory server **should** set the numeric value d associated with each forwarder f in the *Summary* attribute to $d(f) = 1 + P_f$, where P_f is the length of the *Forwarding-Path* attribute for any sufficiently recent forwarder record for f received via the specified neighbor, and the minimum numeric value associated with f across all *Summary* attributes for all directory records received via the specified neighbor otherwise. Similarly, the *Compiled-Metadata* field of the new directory record **should** contain the union of all *Metadata* attributes included in all forwarder records received via the specified neighbor **and** each element of each *Compiled-Metadata* attribute included in each directory record received via the specified neighbor. In this manner, clients **may** be referred to the specified neighbor when they request a forwarder name or metadata field that propagated to this directory server via the specified neighbor.
- NONE. The directory server does not propagate anything received from this peer. This peering directive specifies that a directory server **should** send periodic directory updates to this neighbor but **should not** make use of any directory updates that it receives from this neighbor.

If a directory server is configured such that the final field of some `neighbor` directive takes the form `HOST "." FNAME ".exit:" PORT`, then the directory server **should** wait for the specified neighbor to build a persistent circuit to the directory server before it attempts to establish contact (i.e. request a burst) with that neighbor.

Refer to Figure 9 for an example of how peering arrangements affect propagated records.



directive	records propagated	attributes
full	dir A, fwd A	summary: A1
	dir B, fwd B	summary: B1
	dir C, fwd C	
	dir D, fwd D	
	fwd D1	
	fwd E	
prepend	dir A, fwd A	summary: A1
		fwd-path: E
	dir B, fwd B	summary: B1
		fwd-path: E
	dir C, fwd C	fwd-path: E
	dir D, fwd D	fwd-path: E
	fwd D1	fwd-path: E
	fwd E	
summarize	dir A, fwd A	summary: A1
	dir B, fwd B	summary: B1
	dir C, fwd C	
	dir D, fwd D	summary: D1
	fwd E	
proxy	dir C, fwd C	summary: A,A1,B,B1
	dir D, fwd D	summary: D1
	fwd E	
none	fwd E	

Fig. 9. PEERING ARRANGEMENTS. Consider the scenario illustrated by the diagram shown above the table, in which $\{A, B, C, D, E\}$ are directory servers, with rectangular boxes indicating the peering directives for the indicated neighbors and $\{A_1, B_1, D_1\}$ are standalone forwarders. The table indicates what records are propagated and what corresponding attributes are defined when E applies the indicated peering directives for **both** of its neighbors C and D.

2) Ingress Policy: The `ingress` set of directives provide additional control over route selection, allowing the owner of a directory server to stipulate what records to accept from forwarders and neighbors as well as what preferences to assign to records based upon their propagation paths and the neighbors from which they are received. The directives have the following formats:

```
"ingress pref" SP FNAME SP [FNAME "*"] SP *DIGIT
        "ingress accept" SP [FNAME "*"]
        "ingress reject" SP [FNAME "*"]
  "ingress dir-accept" SP FNAME SP [FNAME "*"]
  "ingress dir-reject" SP FNAME SP [FNAME "*"]
```

Directory servers **should** interpret the configuration directives as follows:

- INGRESS PREF. This directive takes three arguments. The

first argument refers to a specific neighbor, and the second argument refers to the name of a forwarder that might occur in the *Propagation-Path* attribute of a forwarder record or a directory record. The third argument is the score assigned to records received from the specified neighbor, if and only if the second argument is `"*"` or the *Propagation-Path* attribute includes a forwarder that matches the second argument. If two neighbors both propagate either a forwarder record or a directory record corresponding to the same forwarder, then the route selection algorithm uses the `ingress pref` directives to assign scores to both advertisements. The default score for a route (i.e. if it does not match any `ingres pref` directives) is zero. If the scores are unequal, then the route selection algorithm **should** select the route with the higher score. Otherwise, the route selection algorithm considers length of the *Propagation-Path* attribute as described in Section III-C.1.

- INGRESS {ACCEPT, REJECT}. These directives each take a single argument. If the argument is `"*"`, then it matches all forwarders; otherwise, it matches the name of a particular forwarder exactly. If a particular forwarder attempts to publish its forwarder record, then the directory server **should** accept the record if its name matches an `ingress accept` line and reject the record if its name matches an `ingress reject` line. The default behavior is to accept all forwarder records for publication.
- INGRESS {DIR-ACCEPT, DIR-REJECT}. These directives each take two arguments. The first argument refers to a specific neighbor, and the second argument refers to the name of a forwarder that might occur in the *Propagation-Path* attribute of a forwarder record or a directory record (or `"*"`, which matches all records received from the specified neighbor). The directory server **should** accept a record received from the specified neighbor if it matches an `ingress dir-accept` directive and reject a record received from the specified neighbor if it matches an `ingress dir-reject` directive. These directives are considered in advance of `ingress pref` directives.

In all cases, if multiple directives match the same record, then only the first match is considered (order has significance).

3) Egress Policy: The `egress` set of directives do not affect local route selection; instead, they determine which routes are propagated to which peers. The directives take the following formats:

```
"egress accept" SP [FNAME "*"] SP [FNAME "*"]

"egress reject" SP [FNAME "*"] SP [FNAME "*"]
```

Each `egress` directive takes two arguments. The first argument matches the name of a neighbor to which to propagate a record, and the second argument matches the name of a forwarder that might occur in the *Propagation-Path* attribute of a directory record or forwarder record. The directory server **should** propagate a record received to the specified neighbor if it matches an `ingress dir-accept` directive and **should not** propagate a record to the specified neighbor if it matches an `ingress dir-reject` directive. As with `ingress` directives, if multiple directives match the same record, then only the first match is considered (order has significance).

All of the experiments described in the next section have fully open `ingress` and `egress` policy directives.

IV. EVALUATION

To illustrate some of the design tradeoffs inherent to the PAN directory service, we performed empirical measurements using a deployment of roughly 300 nodes on PlanetLab. In our experiments, each of the nodes serves as a forwarder in the PAN overlay, and some subset of the nodes also serve as directory servers. We refer to nodes that perform just forwarding as *standalone forwarders*.

For each of our experiments, we assigned forwarders and directory servers at random from the set of PlanetLab nodes that we had previously determined to be responsive. Our selection process assigns forwarder roles randomly, so the topologies that we chose are *conservative* in the sense that pairs of nodes that directly communicate with each other are determined without regard to the underlying network infrastructure. We suspect that pairwise communicators in most PAN networks deployed in practice would be chosen more intelligently.

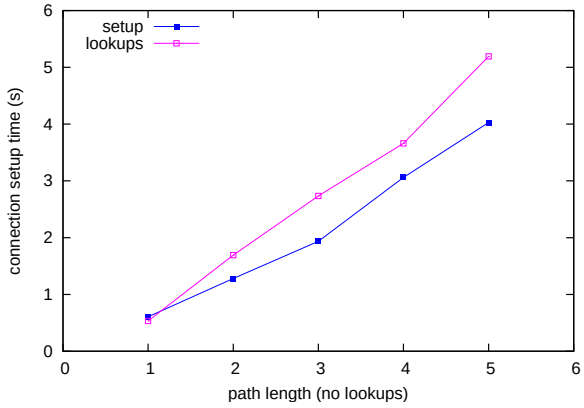


Fig. 10. CIRCUIT SETUP LATENCY. Initial TCP connection setup time increases by an average of 0.8 seconds per hop, and each query issued during the process of constructing a circuit adds an average of about 0.3 seconds.

A. Circuit Setup Performance

To test setup latency for circuits involving multiple hops through the forwarding network and the effect of client queries on TCP connection setup time, we partitioned the nodes into groups of six and assigned each as a directory server (i.e. no standalone forwarders). We used `neighbor` directives to arrange each group into a chain of length five. We then proceeeded to run two tests, as follows:

- GENERIC CIRCUIT-BUILDING TEST. For each group of six directory servers, we used the `prepend` peering directive for all links, directing each directory server to prepend its name to the forwarding path of each forwarder record as it propagates. As a result, the node at one end of the chain has a set of five forwarders to which it can build circuits, each representing a different path length

between one and five. We tested the time taken for Tor to build a circuit for our specified path by requesting to send TCP traffic to some port on the final node in the circuit. The line marked *setup* in Figure 10 shows the average TCP connection setup latency using circuits of various lengths. We are interested in using PAN for interactive applications, and by comparison, user studies have shown that users sometimes shift the focus of their attention after as little as two seconds [23].

- CIRCUIT-BUILDING TEST WITH QUERIES. For each group of six directory servers, we used the `proxy` peering directive for all links, directing each directory server to not propagate forwarder records but instead require the client to issue queries to each successive directory server along the path to the final node in the circuit. As with the previous test, the circuit that the client builds contains between one and five hops, but the client now incurs an additional penalty for spending the round-trip time necessary to perform the query. The line marked *lookups* in Figure 10 shows the average TCP connection setup latency using circuits of various lengths. In each case, the number of queries performed is equal to the number of hops minus one. We observed that the connection setup time increased, relative to the circuits constructeded without queries, by an average of about 0.3 seconds per query.

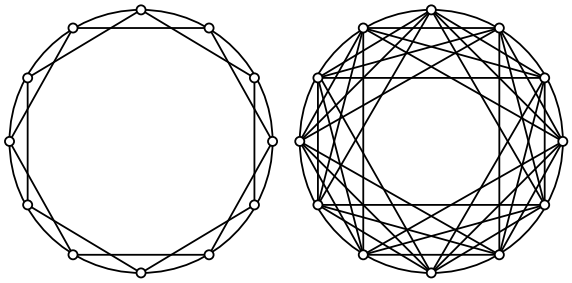


Fig. 11. DIRECTORY TOPOLOGY. In our experiments, we organize the directory servers in a symmetric, circular topology in which all directory servers have the same degree δ.

N	number of nodes ($\sim$ 300)
n_f	standalone forwarders per directory server
n_d	number of directory servers
s_d	size of directory record
$\hat{s}_d$	size of forwarder record with summary
s_f	size of forwarder record ($\sim$ 4 kB)
δ	dirserver connectivity
T_d	directory update interval ($\sim$ 60 s)
T_f	forwarder fetch interval ($\sim$ 600 s)
T_e	forwarder record expiration

Fig. 12. CONTROL PLANE TRAFFIC PARAMETERS.

B. Infrastructure Performance

To evaluate the control plane infrastructure, we generated a number of different topologies by varying a set of parameters; see Figure 12 for a list of the parameters relevant to our infrastructure experiments.

We then performed a series of experiments that involve selecting different combinations of values for T_d, n_f, and δ, as well as different peering directives (specifically, `full` versus `summarize` versus `proxy`). We observed the size and frequency of messages sent between directories and standalone forwarders as well as messages sent among directory servers.

In each case, we used a set of N nodes, selecting n_f standalone forwarders per directory server, leaving $n_d = \lceil N/n_f \rceil$. We organized the standalone forwarders into n_d groups of n_f, such that each forwarder in a group publishes its forwarder record to the same directory server and each directory server receives forwarder records from members of one particular group. Note that as we increase the value of n_f, the number of directory servers decreases, since N is presumed to be constant.

For each experiment, we organized the set of the directory servers into a symmetric, circular topology in which each directory servers has exactly δ neighbors. Forwarders contact their assigned directory servers to publish their forwarder records and download the latest version of the directory every T_f seconds. Directory servers push updates (such as changes to descriptors, withdrawals for forwarders that have failed) to other directory servers every T_d seconds.

Our experiments investigate the following questions:

- What effect does the degree of connectivity, δ, have on the overall amount of traffic on the control plane?
- What effect does the extent of clustering n_f have on the throughput of control messages sent amongst directory servers and between directory servers and standalone forwarders?
- What effect do peering directives `summarize` and `proxy` have on the overall throughput of control messages?
- What effect does the interval T_d between directory updates have on the transfer rate of control messages between directory servers?

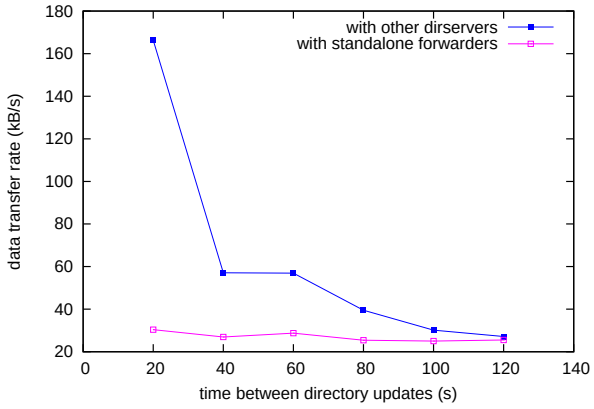


Fig. 13. DIRECTORY UPDATE INTERVAL. Effect of perturbing T_d while setting $\delta = 4$, $n_f = 8$, and peering directive `summarize`.

The following equation governs the data rate r generated by each directory server in the control plane, measured in bytes per second:

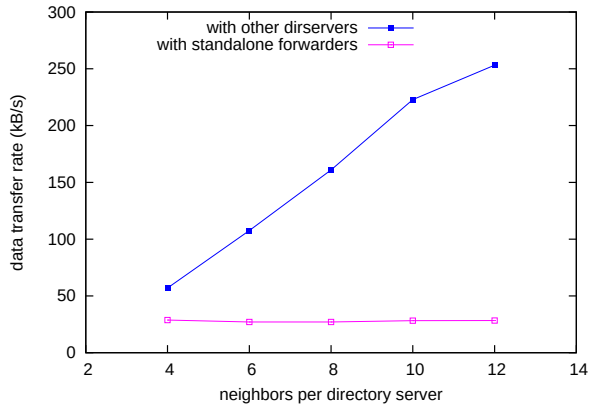


Fig. 14. FORWARDER CONNECTEDNESS. Effect of perturbing δ while setting $T_d = 60$, $n_f = 8$, and peering directive `summarize`.

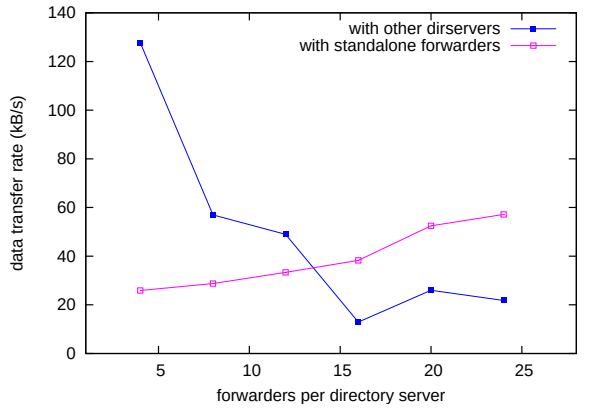


Fig. 15. FORWARDERS PER DIRECTORY SERVER. Effect of perturbing n_f while setting $T_d = 60$, $\delta = 4$, and peering directive `summarize`.

$$r = \frac{n_f u}{T_f} + \frac{\delta u}{T_k} \tag{1}$$

The first term describes the interaction with standalone forwarders, and the second term describes the interaction with neighboring directory servers. The value of T_k is determined by the extent to which the records held by individual directory servers have converged. In an ideal situation, the denominator of the first term would be exactly T_e, though our implementation makes no effort to achieve this goal. The relationships between the various interval values are given by the following expression:

$$T_d \leq T_k \leq T_f \leq T_e \tag{2}$$

The value of u in Equation 1 is determined by the particular peering directive used, as illustrated by the following equations:

$$u_{\texttt{full}} = n_d s_d + N s_f \tag{3}$$

$$u_{\texttt{summarize}} = n_d \hat{s}_d + (n_d + n_f) s_f \tag{4}$$

$$u_{\texttt{proxy}} = \delta \hat{s}_d + n_f s_f \tag{5}$$

Figures 13 through 15 depict the approximate outbound data rate for individual directory servers as observed. The two terms in Equation 1 refer to the two lines in the figures.

Figure 13 illustrates the effect of varying the frequency of updates between directory servers. As the duration between updates increases, the quantity of outbound traffic to other directory servers decreases in inverse proportion to T_d. So, improving the convergence time for the PAN routing tables requires a concomitant investment of bandwidth.

Figure 14 illustrates the effect of varying the number of neighbors to which each directory server is connected. As the number of neighbors increases, the volume of outbound traffic to other directory servers increases linearly, since changes in internal state are propagated to all neighbors. Therefore, improving the robustness of the system by increasing the connectivity between nodes also requires an investment in bandwidth. The figure shows the outcome of an experiment using the `summarize` peering directive, but it is important to note that if the `proxy` peering directive were used instead, then the volume of control plane messages would still increase proportionally with δ, but circuit setup time would decrease with δ, since the number of lookups required to discover the circuit to be created through the overlay would decrease.

Figure 15 illustrates the effect of varying the number of standalone forwarders that publish their forwarder records to a given directory server. Since our experiments use a constant number of nodes, adjusting this parameter changes the ratio of directory servers to standalone forwarders. Specifically, n_f increases while n_d decreases. Since we are using the `summarize` peering directive, the volume of traffic between a given directory server and standalone forwarders increases because n_f dominates the first term of Equation 1, whereas the volume of traffic sent to other directory servers decreases because n_d dominates the second term of Equation 1. So, increasing the number of "leaves" in the topology by decreasing the ratio of directory servers to standalone forwarders alleviates some of the traffic in the core of the network but increases traffic at the edges. Robustness is not necessarily affected, since forwarders can publish their forwarder records to multiple directory servers. While we do not show experimental results for that situation, we assert that directing each standalone forwarder to publish its forwarder record to m directory servers involves substituting mn_f for n_f in Equations 1 through 5.

C. Traffic Profiles

Figures 16 and 17 depict the average outbound traffic volume per minute for a typical directory server. Figure 16 presents the outbound traffic between a directory server and its neighbors, given peering rule `summarize` and two different values of n_f. Observe that the traffic volume levels off after increasing for the first twenty minutes while PlanetLab nodes come online and routing information converges. Figure 17 shows the average outbound traffic volume per minute to standalone forwarders. The periodicity is the result of periodic directory fetches at time interval T_f on the part of standalone forwarders.

In Figure 18, we show the overall traffic volume of control messages sent between directory servers and standalone for-

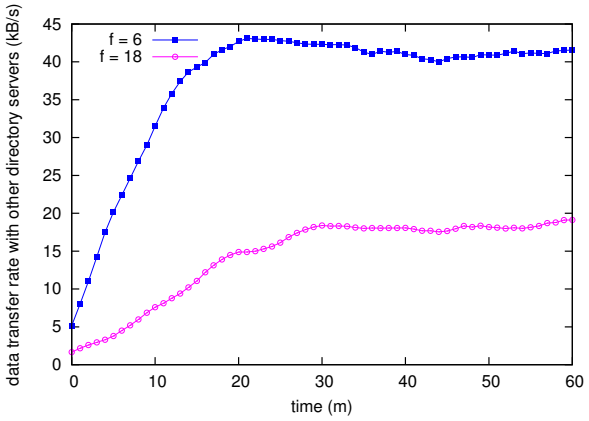


Fig. 16. INTER-DIRECTORY TRAFFIC PROFILE. Five-minute moving average snapshots, by minute, for traffic from a typical directory server to its neighbors, given $n_f = 6$ and $n_f = 18$. We define $T_d = 20$, $\delta = 8$, and peering directive `summarize`.

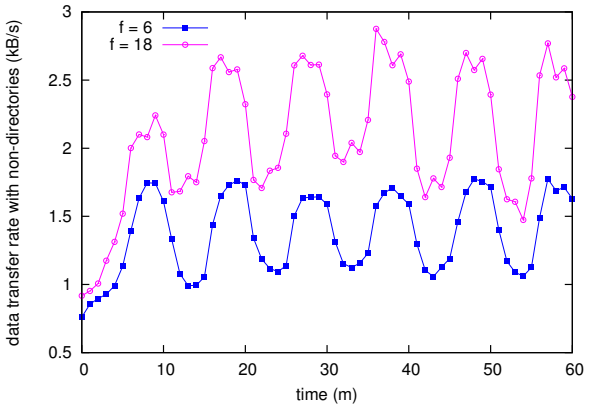


Fig. 17. TRAFFIC PROFILE BETWEEN A DIRECTORY SERVER AND STANDALONE FORWARDERS. Five-minute moving average snapshots, by minute, for traffic from a typical directory server to the forwarders whose forwarder records are published directly, given $n_f = 6$ and $n_f = 18$. We define $T_d = 20$, $\delta = 8$, and peering directive `summarize`.

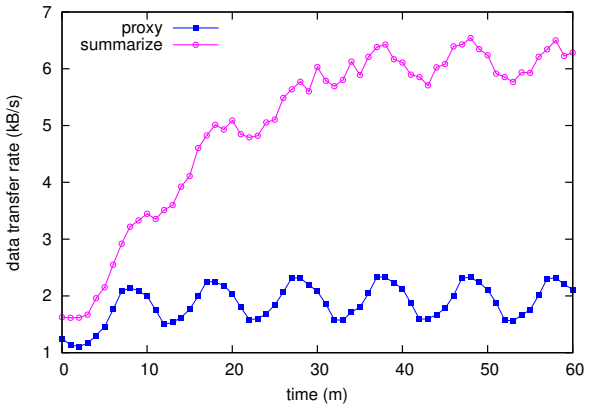


Fig. 18. TRAFFIC PROFILE: PROXY VERSUS SUMMARIZE. Five-minute moving average snapshots, by minute, for traffic from a typical directory server to the forwarders whose forwarder records are published directly, given $n_f = 18$, $\delta = 4$, and $T_d = 20$, using peering directives `proxy` and `summarize`, respectively.

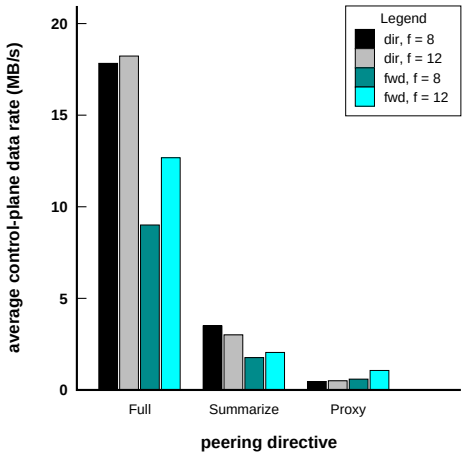


Fig. 19. PEERING DIRECTIVE COMPARISON. Effect of peering directive on traffic volume. We show examples for $n_f = 8$ and $n_f = 12$, given $\delta = 4$ and $T_d = 60$. Bars marked `dir` indicate traffic to neighboring directory servers; bars marked `fwd` indicate traffic to forwarders.

warders using peering rules `proxy` and `summarize`. Observe that `proxy` exhibits signifcantly lower transfer rates because directory servers need not provide forwarder records for every directory server in the network. Instead, clients are referred to successive neighbors at each query in which a forwarder record is not found. We observe an inherent tradeoff between circuit performance and traffic volume to standalone forwarders, as described in Section IV-B. A network designer would consider this effect in selecting a peering directive.

Finally, Figure 19 presents a summary of how peering directives affect control plane activity. We conclude that peering directive `full` is probably too expensive to justify the decrease in circuit setup latency for the general case, while peering directive `proxy` reduces control plane traffic quite substantially, but at a cost to circuit performance that may be prohibitive. Which peering directive to choose is inevitably a function of the constraints of the underlying network topology and the needs of client applications.

V. CONCLUSION

We have presented the design of a directory service for use with Perspective Access Networks, including a specification of the architecture, a discussion of the design tradeoffs, and an evaluation of the performance using our working implementation.

Future work includes addressing a number of interesting technical questions that examine how PAN interoperates with environments where there is deliberate restriction of access to resources, such as governments censoring the web sites that their citizens could otherwise view. In such a scenario, we need to study how effectively PAN could provide access to blocked resources despite continual discovery and shutdown of PAN forwarders that enable this access. Second, we need to design mechanisms for performing web searches across fragments.

The locality feature of PAN could be used to improve web searches in the Internet today as well as in the increasingly fragmented Internet of the future.

Equally interesting are the policy questions that arise from having a system like PAN deployed across the Internet. Many enterprises use end-to-end authentication for some of the services they provide in their private networks, but there are a number of popular services that rely upon the assumption that the only hosts that have access to the service are physically on the same LAN or have particular network-layer addresses. Moreover, deployment of PANs in the Internet could threaten the business models of companies providing or depending on geolocation services for anti-fraud resolution, digital rights management, and spam detection. Convincing these parties to move away from network-layer authentication as the basis for their security will be an interesting task.

Our design of PAN has been motivated by four main objectives: (a) to enable perspective sharing across the Internet by providing universal access to resources, (b) to allow locality in naming, (c) to demonstrate ease of deployment, and (d) to promote decentralized management. Our design decision not to impose a global naming scheme of resources helps us achieve these objectives but bears the cost of sacrificing aggregation. Nonetheless, with recent new threats to Internet consistency (governance disputes, geolocation services, and accidental or deliberate censorship of resources), it is worth considering the implications of an Internet without a well-defined core, consisting of fragments whose names and address spaces are not ordained hierarchically. Our work in building a Perspective Access Network is a step in this direction, and our directory service architecture represents the core of this effort.

VI. Acknowledgements

We would like to thank the following individuals whose insightful comments have greatly benefited our project: H.T. Kung, David Parkes, Roger Dingledine, and Nick Mathewson.

References

[1] A. Chankhunthood, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarhical Internet Object Cache. In *USENIX*, Jan. 1996.

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

[3] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the Addressing Architecture. *ACM SIGCOMM Computer Communication Review*, pages 313–321, 2003.

[4] D. Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing Reality: An Architectural Response to Real-World Demands on the Evolving Internet. In *Proceedings of ACM SIGCOMM 2003 FDNA Workshop*, 2003.

[5] E. Cohen and H. Kaplan. Aging Through Cascaded Caches: Performance Issues in the Distribution of Web Content. In *Sigcomm*, 2001.

[6] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. Internet Engineering Task Force: RFC 2234, 1997.

[7] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. Plutarch: an Argument for Network Pluralism. *ACM SIGCOMM Computer Communication Review*, 33(4):258–266, 2003.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the Seventh USENIX Security Symposium*, 2004.

[9] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A scalable Wide-area Web Cache Sharing Protocol. In *SIGCOMM*, 1998.

[10] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Censorship and Surveillance. In *Proceedings of the 11th USENIX Security Symposium*, 2002.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol: HTTP/1.1. Internet Engineering Task Force: RFC 2616, 1999.

[12] H. W. French. Despite Web Crackdown, Prevailing Winds Are Free. *The New York Times*, 9 February 2006.

[13] Intel Corporation. The Evolution of the Next-Generation Internet. `http://www.planet-lab.org/`, 2003.

[14] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *STOC*, 1997.

[15] H. Lewis. Online Fraud Catchers: Protecting You but Maybe Also Getting Your Card Turned Down. `http://www.intelligentbanking.com/brm/news/ob/20000915.asp`, Dec. 2002.

[16] J. Menaud, V. Issarny, and M. Banatre. A New Protocol for Efficient Transversal Web Caching. In *Symposium on Distributed Computing*, 1998.

[17] P. Mockapetris. Domain Names: Concepts and Facilities. Internet Engineering Task Force: RFC 1034, 1987.

[18] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proceedings of ACM SIGCOMM*, 1987.

[19] T. S. E. Ng, I. Stoica, and H. Zhang. A Waypoint Service Approach to Connect Heterogeneous Internet Address Spaces. In *Proceedings of the USENIX Annual Technical Conference 2001*, 2001.

[20] Open Net Initiative. Internet Filtering in Iran 2004-2005. Open Net Initiative Case Study, June 2005.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM)*, pages 161–172, 2001.

[22] C. Rhoads. Endangered Domain. *The Wall Street Journal*, 19 January 2006.

[23] V. Roto and A. Oulasvirta. Need for Non-Visual Feedback with Long Response Times in Mobile HCI. In *In Proceedings of WWW2005*, May 2005.

[24] A. C. Snoeren and B. Raghavan. Decoupling Policy from Mechanism in Internet Routing. In *Proceedings of the Second Workshop on Hot Topics in Networks*, 2003.

[25] J. Stewart. BGP4: Interdomain Routing in the Internet. Addison-Wesley, 1998.

[26] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM*, 2002.

[27] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[28] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *ICDCS*, 1999.

[29] J. Touch. The LSAM Proxy Cache - A Multicast Distributed Virtual Cache. In *3rd WWW Caching Workshop*, June 1998.

[30] V. Valloppilli and K. Ross. Cache Array Routing Protocol v1.0. IETF draft-vinod-carp-v1-02.txt, Feb. 1998.

[31] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[32] D. Wessels. Squid Internet Object Cache. `http://www.squid-cache.org/`, 2001.

[33] T. Wright. EU Tries to Unblock Internet Impasse. *The New York Times*, 30 September 2005.