Separating Routing and Forwarding: A Clean-Slate Network Layer Design (Invited Paper)

Kenneth L. Calvert, James Griffioen and Leonid Poutievski
Laboratory for Advanced Networking
University of Kentucky
{calvert,griff,leon}@netlab.uky.edu

***Abstract*— We present a "clean-slate" design for a network-layer routing and forwarding system intended to address shortcomings of the current Internet Protocol. Our design separates routing from both forwarding and topology discovery; requires only a flat, topology-independent namespace; and allows for policies of both users and service providers to be supported. *Channels* serve as the primary abstraction, allowing the network topology to be viewed at multiple levels of abstraction using the same identifiers. In this paper we present the basic design, which is based on loose source routing. Our routing and forwarding scheme is part of a larger project to produce a "clean-slate" network layer design.**

I. Introduction

Although the Internet protocol suite has been amazingly successful at supporting new applications and services, many of the original assumptions underlying its design are no longer valid. There is increasing agreement in the community (including funding agencies) about the need to explore clean-slate designs. Somewhat surprisingly, however, few proposals for new network layer designs have been put forward to date. Instead, researchers generally focus on particular aspects that are considered problematic in today's Internet, such as interdomain routing policy, reducing unwanted traffic, optimizing resource usage, etc. (By "network layer", we mean the protocols that govern the end-to-end delivery of information, which are expected to be implemented by the participants in the network—in other words, the "waist of the hourglass". By "architecture" we mean the set of functions implemented by the system, and how they are assigned to the various components.) In this paper, we describe a novel routing and forwarding architecture that is part of a larger project to develop a clean-slate network layer design. The larger project is called *Postmodern Internetwork Architecture* [4]; the forwarding/routing approach presented here is called *postmodern forwarding and routing infrastructure* (PFRI).

Our work has several high-level goals. First, we want the functionality of the forwarding infrastructure—the network elements that *must* exist to interconnect channels and form the network—to be independent of most aspects of the end-to-end service. In particular, *it should be possible for the end-to-end service to evolve without a "forklift upgrade"*. A network layer that could support different end-to-end services was in some respects a goal of the current Internet architecture [10], but certain aspects are nevertheless entangled in IP—specifically routing, forwarding, and addressing. (Consider the difficulties involved with the transition to IPv6, which was essentially nothing more than a change of numbering scheme.)

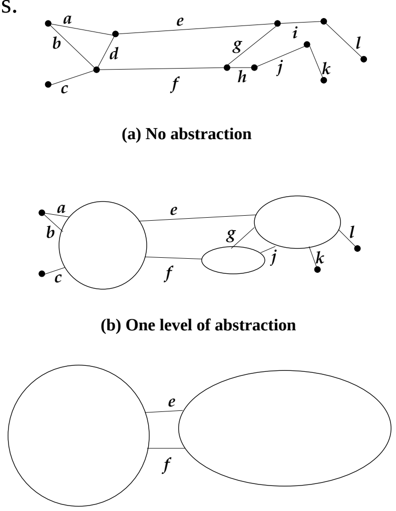
Second, our design is intended to *recognize and isolate the distinct and sometimes contradictory interests* of the users who generate the traffic and pay the bills, and the service providers who carry the traffic and collect those bills. The present Internet architecture's lack of mechanisms to support different parties' policies has led to "tussles" [9], and to many hacks becoming practically indispensible for the continued operation of the network. (Consider the use of IP addresses and—especially—port numbers to determine whether traffic can be allowed into a domain.)

Third, our design should be *flexible with respect to where and how often functions are performed*. For example, we want to be able to push functionality into the end systems and/or special infrastructure (a la DNS). In the present Internet, there is actually very little of this kind of flexibility: some basic responsibilities, such as end-to-end reliability, are delegated to end systems, but others, such as route discovery and selection, must be implemented in virtually every router. Along the same lines, we want to allow a greater range of options for the frequency of occurrence of potentially costly operations. In the current Internet, IP's hop-by-hop routing/forwarding function treats all destinations the same, expending the same effort to find routes to *all* destinations, regardless of the traffic's actual mix of destinations. In other words, IP spreads the cost of finding all routes over all packets equally; in a world where the cost to send a packet may vary widely across sources, it may be useful to allow a wider variety of amortization schedules. (Consider a battery-powered handset on the edge of radio range versus a large server in a corporate data center.)

The focus of this paper is on a basic datagram delivery service, which has the following features:

- **Separation of routing and forwarding.** This is also a feature of switched, connection-oriented networks like ATM, of course. Unlike switched networks, however, our design does not require per-flow state in the forwarding infrastructure, although it is permitted as a performance optimization. Instead, each packet carries a *forwarding directive*, which partly specifies the path it should follow through the network. The full path followed by the packet is determined by the policies of the source and the providers whose networks carry the packet.

- **Separation of topology discovery and path selection.** At all levels, the problem of topology discovery (i.e. discovering potential routes) is separated from route-selection policies that are likely to be determined by business relationships.
- **Flat, topology-independent identifiers.** PFRI has no naming or addressing hierarchy, and no central naming authority is required. Identifiers that are required to be globally unique for routing/forwarding are selected randomly or pseudorandomly, from a large space, to (virtually) guarantee uniqueness. This separation makes it easier to support mobility.
- **Flexible topology abstraction:** In PFRI, *only channels are named* (i.e., are assigned identifiers); nodes remain anonymous. The advantage of this approach is that the network can be viewed at different levels of abstraction using the same channel names (see Figure 1). Where necessary, nodes can be implicitly identified with the set of channels they terminate, including abstracted clusters of nodes.



(a) No abstraction

(b) One level of abstraction

(c) Two levels of abstraction

Fig. 1. Channel names do not change regardless of abstraction level, while different node names would be required at different levels of abstraction.

Although our primary focus here is on the routing/forwarding approach, it is only one component of the network layer being developed in the Postmodern Internetwork Architecture project. Because the other components are crucial to the viability of PFRI, we discuss their role here briefly. (Indeed, we believe the lack of interest in prior source routing proposals, including some that PFRI resembles, is due at least in part to the absence of these functions.) These additional functional components, each representing a part of the internetwork-layer header, are [4]: *motivation*, *accountability*, *knobs*, and *dials*. We consider each in turn.

In today's Internet, packets flow between domains only when either a customer-provider relationship or a peering relationship exists. Because routing in IP is done hop-by-hop, customers are completely dependent on their immediate upstream provider to determine the route followed by the packets they send, even though that route generally involves service providers other than the originator's. The purpose of the *motivation* field is to remove this limitation, so that customer-provider relationships can exist apart from topology. The idea is that nodes in the infrastructure, before forwarding a packet, check its motivation field for a reason to do so. Thus the motivation field might contain credentials that prove the originator is a legitimate member of the local domain, or is a customer of a particular transit provider. Recently other researchers have proposed the use of *network capabilities* for similar purposes [14], [16], [17].

The *accountability* mechanism provides a hard-to-forge record of a packet's origin and handling in the network. Being able to attribute packets to a specific endpoint and path has many uses, including tracking malicious packets and using flooding as a basic functional building block.

Knobs enable packet sources to supply *advice* to the network, in order to increase the likelihood of adequate performance in some dimension. An example would be an indication that a packet is the first of a series of packets, all going the same way. Such a hint would enable the infrastructure to, for example, establish state in order to amortize resolution or lookup costs over all of the packets.

Dials support information flow in the other direction, i.e. from the network to the user. An example would be a signed, nonrepudiable timestamp indicating the time of arrival at each hop along the path specified by the source. Such information would enable a user to determine where delay builds up in the network.

The rest of this paper is organized as follows. The next section introduces the concepts and terminology of PFRI. Section III discusses the model of usage of the network and compares it to the current Internet, particularly with respect to the parties responsible for various functions. Section IV describes the basic operation first in the context of a simplified model, then shows how the approach can be made to scale. In Section V we discuss some additional considerations related to performance and other details of operation. In Section VI we discuss some connections and differences between our work and others'. Section VII concludes the paper.

II. Concepts and Terminology

A *channel* is a logical means of transmitting packets from one location to another. In our architecture, it may correspond to an actual physical channel (such as a SONET link, a lambda in a fiber, or a wireless LAN), or to a higher-level abstraction such as a buffer queue in an operating system, a label-switched path, or a pair of IP addresses in the existing Internet.

Each channel in the network is identified by a globally unique, unstructured *channel ID*. As noted in the introduction, channel IDs are chosen (pseudo)randomly from a large space to avoid the need for any central naming authority. Although we do not make use of it in this paper, each channel ID is assumed to be bound to a public/private key pair in a self-certifying way (for example, the channel ID is the hash of the public key). These considerations make it necessary for channel IDs to be rather large (at least 64 bits). We therefore note here a fundamental premise of our design is that *header bits are cheap*. The PFRI header, which in general will contain several channel IDs, can be rather large—perhaps hundreds of bytes. Our position is that in today's network, bandwidth is plentiful, and it is more important to include required mechanisms than to optimize bandwidth consumption by headers *throughout* the network architecture. In those places where bandwidth really is precious, state and other methods can be used to shrink the overhead [11]. Moreover, increasing the minimum packet size provides a benefit: a corresponding increase in the per-packet processing time budget.

For the purposes of this paper, all channels are understood to be point-to-point and bidirectional. While in practice some link layers do not have these attributes, it is clear that, for example, point-to-point channels may be built upon multipoint link layers. Although unidirectional links may occasionally be encountered (for example in mobile/wireless networks), they are of limited utility in the context of a general-purpose network unless other, reverse channels exist. We therefore believe the conceptual simplification of this assumption is worth the burden it places on underlying link layers.

A *node* is the logical endpoint of one or more channels. It may represent a program running in a host, a piece of hardware (e.g. a sensor), a server farm, or an entire network domain such as a university campus. A *forwarding node (FN)* is a node that provides "transit service", i.e. relays packets among its attached channels. Although a FN could be a piece of hardware infrastructure (like a router or switch), it might also be a larger component, such as a complete transit domain (like an AS in today's network).

The term *endpoint* refers to a node that acts as a source or sink of traffic (packets). An endpoint will typically be a process or program, but might also be a file, port, mailbox, etc. Each endpoint has one or more *end-channels* through which it can send and receive packets. An end-channel is any channel that terminates on an endpoint. We use the term EID to refer to the channel ID of an end-channel, although there is no apparent difference between the ID of a regular channel and an end-channel.

A *realm* is a collection of channels and nodes that are under a common administration at some level. Realms may be nested hierarchically. In our architecture, a realm is indistinguishable from a node when viewed from the outside. In particular, some realms provide transit service between channels, while others do not. Realms are a generic construct corresponding to the various levels of hierarchy in the present network: autonomous systems, OSPF areas, subnetworks, etc.

A *path* is a sequence of channels, where adjacent channels in the sequence have a common endpoint, which is a FN. A *partial path* is a sequence of channels that does not necessarily have the connectivity property, i.e. consecutive channels in the sequence are not necessarily connected by a FN. A *forwarding directive* is a component of the PFRI header containing a partial path, an indication of the current location in that path, and information that distinguishes among data traffic and various forms of signaling traffic.

III. PFRI Identifiers and Usage

The forwarding mechanism resides at the heart of the PFRI architecture, and uses a very simple, flat namespace. As in the current Internet, other namespaces will also be needed, as will means of mapping names from the various spaces to each other. Separation of concerns with respect to any auxiliary infrastructure required to perform these mappings is also important.

The following describes our vision of the use of various names and identifiers in PFRI, including who controls/supplies the resolution mechanisms and sets policy. Fig. 2 shows the various objects used in establishing communication (boxes). Arrows indicate mappings. For comparison purposes, we also describe the corresponding mapping steps for the current system.

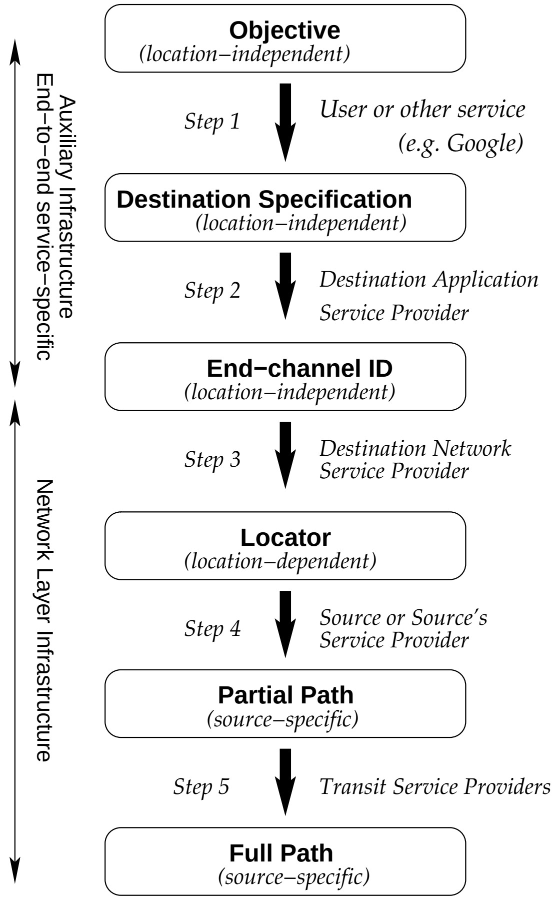


Fig. 2. Resolution steps; next to each arrow is the entity responsible for that mapping

1) The *communication objective* represents what the user wants to communicate with. It generally remains implicit, both today and in PFRI. It is typically resolved into a *destination specification*. In today's network, the destination specification is partly explicit and partly implicit. Examples of the explicit part would be a URL, an email address, or a DNS name. The implicit part is the well-known port associated with the application, which is typically built into the application. This "resolution" is often performed manually (for example, the user knows the email address of the person they want to contact) or through a third-party service such as Google. In any case, the destination specification identifies a particular service.
2) The *destination specification* is resolved to an *EID*, which uniquely identifies the endpoint of communication. This resolution step, which represents the selection of a specific *instance* of a service, is controlled by the destination (application) service's administration. In the current Internet, the EID is an IP address plus (implicitly) a port number and protocol; the resolution is typically done via DNS. The provider of the (application) service has control over this mapping, both today and in PFRI. Because the EID today is tied to location, the provider may return a different value depending on the requestor's location (cf. Akamai). In PFRI, on the other hand, the EID is the (opaque) channel ID of an end-channel, which identifies an *application entity* independent of its location.
3) The *EID* maps to a *locator*, which encodes information about the endpoint's location in the network. In the present Internet this step does not exist, because the IP address encodes the endpoint's location and acts as *both* EID and locator. The drawbacks of this approach are well known. In PFRI, this resolution is handled by a dedicated, hierarchical mapping service, and the destination network provider's policy determines the mapping. (Details are discussed in Section IV-D.)
4) The *locator* is elaborated to a *partial path*. In PFRI, this mapping is performed by the source and the source's network service provider (or by the latter exclusively), according to their respective policies; when the partial path is obtained the packet can be handed off to the PFRI layer.
5) The *partial path* is refined to a *complete path* by filling in gaps, which correspond to intra-realm segments. In PFRI, this step is controlled by the realms (transit providers) through which the packet passes.

In today's Internet, the packet is passed to the network layer after step 2, and the mapping from locator (IP address) to complete path—is performed hop-by-hop as the packet travels through the network. Each node determines the next hop in the path according to *its own policies* and the destination IP address.

The first two resolution steps are considered to be auxiliary functionality that uses the network layer. Like the DNS in today's network, they are useful but, strictly speaking, not necessary for its functioning. The PFRI network layer enters the picture only when the objective has been resolved to an EID. Before taking a closer look at these network layer mechanisms and how they can be made to scale, we point out that the separation of concerns outlined above is quite consistent with that of the three-layer name system proposed by Balakrishnan et al [3].

IV. Forwarding and Routing Operation

The fundamental service provided by the PFRI layer infrastructure is relaying a packet from channel to channel, as specified by a forwarding directive carried in the packet. The FD specifies a sequence of channel IDs; as long as the sequence represents an actual path in the network, the basic operation is straightforward: When a packet arrives at a FN, the "current position" indicator in the FD is updated, the next channel ID in the sequence is mapped to one of the FN's attached channels, and the packet is forwarded over that link. The challenge is to enable the FD to be constructed in the first place. Our design divides the responsibility for this problem among the source and the various providers who will carry the packet.

We first describe how PFRI works based on certain simplifying assumptions. Then we show how to relax the assumptions *without depending on any routing mechanism*. In what follows, we mostly ignore the other functional blocks of the architecture (motivation, accountability, etc.), except to point out where their functionality may be useful.

For this simplified scenario, we assume the network has only a single level of hierarchy: Each channel either connects two nodes within the same realm, in which case it is called an *interior channel*, or it crosses into a different realm, in which case it is called a *border channel* of the realm. A node connected to a border channel is called a *border node* of the realm. Some realms act like forwarding nodes, i.e. they relay packets among their connected channels, while others do not.

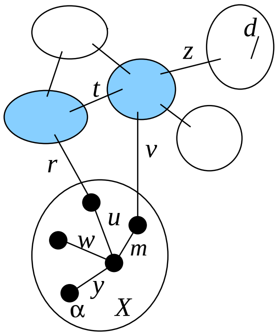


Fig. 3. Network graph for simplified scenario

We assume the following:

P1: Each node in each realm knows the internal topology of its local realm, including the identifiers of every internal and border channel of that realm.

P2: Each node knows the (connected) realm-level topology, i.e. the connectivity and identities of all inter-realm channels in the network.

P3: There exists an EID-to-border channel mapping service. Given the channel ID of any end-channel in the network, the service returns one or more border channels of the realm that contains that end-channel. Every realm contains at least one provider of this service, and every node in the realm knows the identity of that provider.

Consider a node (call it α) in a realm X, which has a packet destined for an application identified with with end-channel d.[1] (Refer to Fig. 3, in which realms that provide transit service are shaded.)

The node α does not know the location in the topology of the application associated with EID d, so it must resolve d to something it does know. It sends an *EID resolution request* packet to its local EID-to-border channel service; the forwarding directive of the PFRI header of that packet contains the path yw. (The PFRI header must also contain, in its *motivation* field, credentials to convince the common endpoint of y and w to forward the packet.) The EID-to-border channel service receives the request, looks up d and returns z as a border channel of the realm containing d. The packet is returned to α using the reversed FD wy.

Now α can use its knowledge of the realm-level topology to select an inter-realm path that leads to z according to its own path-selection policy. Since the realm containing d is not a neighbor of X, one or more realms will need to provide a *transit service* to the packet. If α is an established customer of one of the candidate transit realms, it may select a path that uses one of those realms; otherwise, it must establish a relationship with some transit provider(s) to get the motivation credentials needed to convince the realms to provide transit service. (For now, we assume this transaction happens out-of-band.) In our example, α selects the inter-realm path rtz.

Using its knowledge of the local topology, α selects the *egress* portion of the path to r. In our example yu is the only choice, but in general α may apply policy here to select a path. So α constructs a FD containing the partial path $yurtzd$. (It also constructs and includes the needed motivation credentials for the channels included in this path.) It sends the packet over channel y. The packet is forwarded over y, u, and r, with the FD being updated at each hop as described above. (In addition, the packet's motivation is also checked at each hop.)

After traversing channel r, the next channel in the sequence is *not* directly attached to the receiving node: the FD does not specify the pathto be followed through the first transit realm. At this point, we say a *path fault* occurs. We discuss the handling of path faults later, in Section IV-B; for now, we simply state that the transit realms are responsible for conveying the packet from r to t, and from t to z. (Given P1, it should be clear that each realm can construct a path connecting the border channels.)

After the packet crosses z, another path fault may occur. Again, given P1, it can be handled and the packet is conveyed across d. At this point the network layer has fulfilled its obligation, i.e. the packet was transmitted and received over each channel in the FD. If the end-channel associated with the EID corresponds to the input queue of an application, no further demultiplexing is necessary. Alternatively, the end-channel might be the input of a module that will perform additional protocol processing. (Interaction with higher-level protocols is discussed in Section V.)

A. *Implementing P1–P3*

Next we consider how the assumptions can be made to hold without begging the question, i.e. without relying on any other routing mechanism. The first two are straightforward; P3 is a bit trickier.

Regarding **P1**: Knowledge of the intra-realm topology can be obtained and propagated using a link-state-like protocol along the lines of OSPF [12]. Each node in the realm periodically creates and sends a *topology advertisement* that lists all of its channels. These advertisement messages are *flooded* throughout the realm. (Note that the accountability mechanism makes it relatively easy to implement a safe and efficient flooding primitive.) Topology advertisements come in two flavors: regular and transit. Regular advertisements indicate that a node can be reached via the advertised channel(s). Transit advertisements imply that in addition the originating node is prepared to *forward* packets between any of the advertised channels (given adequate motivation). Thus the advertisement emitted by each forwarding node is a transit advertisement; hosts emit regular advertisements.

Regarding **P2:** Given P1, the same basic method can be used to distribute information about the realm-level topology. Here, the border nodes in each realm play a special role as *aggregators* of the realm-level topology information. Using their knowledge of the internal and border channels of the realm, they construct an advertisement of the realm's border channels and forward it to each of their neighbors in other realms. Those neighbors, in turn, flood the advertisement throughout their realms and pass it on to neighboring realms. Eventually each node in the network receives an advertisement for each realm, and has a complete graph model in which realms are nodes and inter-realm channels are the edges.

Regarding **P3:** given P1 and P2, we can construct a global service that maps EIDs to the border channel IDs of the realms that contain them. In each realm, one or more nodes is designated to provide this service. (This designation must be part of the node's configuration.) These nodes periodically flood announcements of their location throughout the realm; in this way each node in the realm learns a path to the service. Moreover, border nodes forward these announcements to their neighbors in adjacent realms, which forward them directly to their own designated service nodes. In this way, the service nodes in each realm learn of their counterparts' locations.

The EID-to-border channel service nodes in each realm organize themselves into a global DHT using their knowl-

[1] We name nodes and realms in this example for convenience only; the architecture does not require that they be named.

edge of the realm-level topology as well as their own intra-realmtopology. First, each service node floods an announcement of its existence throughout its realm; this enables all other nodes in the realm to learn a path to it. The border nodes of the realm exchange this path information across the realm boundaries, so that service nodes in neighboring realms learn paths to each other. The service nodes then organize themselves into a distributed hash table (DHT) structure such as a Chord ring [15]. For Chord, each service node chooses a neighboring service node as a successor. Using their knowledge of the topology, nodes then identify the nodes that are 1/2, 1/4, etc. of the way around the ring, and construct paths to those *finger* nodes; thanks to P1 and P2, these paths can be made efficient in terms of the underlying network graph. Finally, the service nodes partition the identifier space so that each is responsible for an approximately equal-sized portion of the space; each node then inserts the mappings for its realm into the DHT. (In practice, the global DHT would probably not bootstrap itself in this way, but would be administratively configured to provide efficient service.)

B. Path Fault Handling

As noted above, a FD may contain gaps. One reason is scalability: it is not reasonable to expect a source to know enough about the network topology to specify a complete path to any destination. Another is privacy: realms may not wish to expose information about their internal topology outside the realm. For these reasons, path faults may happen during forwarding.

When a packet arrives at a forwarding node and the next channel in the FD is unknown, a *path fault handler* (PFHs) is invoked; the PFH may or may not be co-located with the border node at which the fault occurred. If it is not, the FN must be configured with a FD to get the packet to reach the PFH; this FD is placed in a new PFRI header (which flags the packet is being in the midst of path fault handling), which is "pushed" (prepended) onto the packet, which is then forwarded to the PFH, where the temporary header is popped off again by the PFH.

When a PFH receives a "path fault" packet from a faulting node, it can resolve the fault in any of several ways. It might, for example, determine a path from the faulting FN to the target egress channel. In this case, it constructs a FD and PFRI header containing the "gap-filling" intra-realm segment and pushes it on the packet, followed by another header containing the reverse of the FD just popped. The packet is forwarded back to the FN where the fault occurred, the outer header is popped, and then the packet is forwarded to the egress channel, where the gap-filling header is popped, and the packet continues on its merry way. This approach is inefficient, but allows the PFH to direct the faulting node to *cache* the gap-filling header (including any motivation information required to get the packet through the realm).

The alternative approach is to push onto the packet a path from the PFH directly to the target egress channel. The packet is *not* returned to the faulting FN, and no cache entry is added at the faulting FN. As a result, future faults will also be forwarded through the PFHon the way to the egress channel. This approach would only be used if the packet indicates (via the *knobs* field) that it is one-of-a-kind, and no state should be established. Thus, when selecting a route for the packet to follow, the PFH examines not only the *FD* and *motivation* fields, but also and the *knobs* field in the packet header. The latter may indicate particular characteristics desired for the transit path, especially if the packet is the first of many to flow along the same path. Realms that provide transit service will often have multiple flows traversing the realm through the same ingress/egress channels. Although the flows share the same ingress and egress channels, they may have differing QoS requirements (knobs) and must be routed over different paths inside the realm. By looking at the knobs of every faulting packet, the PFH can select an appropriate one. To avoid routing all packets through the path fault handler, it is necessary to allow paths in a FN's path cache to be associated with a *flow ID* as well as a channel ID. The ability to insert path cache entries that define different routes to the same egress channel gives PFHs the ability to support QoS routing and traffic engineering while amoratizing the costs of route faults/computation across many packets in a flow.

C. Hierarchical Topology

Clearly, for this system to scale to millions or billions of nodes, it is necessary to support multiple levels of abstraction (hierarchy). However, we want to avoid placing arbitrary limits on the *amount* of hierarchy that can be supported, so we allow for arbitrary nesting of realms. That is, any realm may contain any number of sub-realms, some of which may contain sub-sub-realms, etc. Thus we must revise P1 and P2:

P1a: Each node knows the identity and internal connectivity of all interior and border channelsof every realm that contains the node.

P2a: Each node knows the complete top-level inter-realm topology, i.e. the identities and connectivity of all channels that are not interior to any realm.

The possibility of different amounts of hierarchy in different parts of the network complicates the propagation of topology information needed to achieve these properties. In the description above, border nodes need only consider whether a channel crosses a realm boundary or not to determine what information to include in advertisements sent over that channel. In particular, advertisements sent over a border channel contain *only* the border channels of the originating realm(plus an indication of whether transit service is offered among them). Advertisements *received* over a border channel can likewise be assumed to contain only inter-realm channels. However, with different levels of nesting in different places, this is no longer true. Fig. 4 shows an example.

In the figure, nodes are indicated by shaded circles (and labeled for convenience with lower-case Greek letters); ellipses

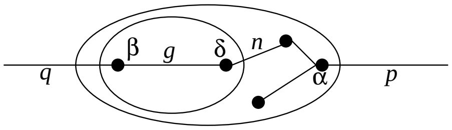


Fig. 4. Unbalanced nesting of realms

denote realm boundaries.[2] Consider node β, which will receive an advertisement from node δ over channel g. According to the algorithm given above, that message will list two links, g and n. Since n is a border channel, according to the algorithm β should include it in the advertisement it emits over channel q. However, n is actually completely contained within the outer realm boundary crossed by q; therefore β should not include it in the advertisement, but *should* include channel p, which crosses the same *outer* realm boundary as q. Similarly, α should emit an advertisement over p that contains both p and q.

To achieve this, we first introduce a notion of the *level* of a channel. In order to define channel level, we first define the level of a *realm* to be the maximum depth of nesting within that realm; that is, the maximum, over all nodes in the realm, of the number of realms (within in the original realm) that contain the node. This corresponds to intuitive notions of the amount of hierarchy within a realm. (For example, the level of the outermost realm in Fig. 4 is two.) Now we can define the *level of a channel* h *at a node* μ to be the maximum level of all realms containing μ whose boundary is crossed by h. For example, the level of both p and q in Fig. 4 is two, while the levels of g and n are zero and one, respectively. Finally, we define the level of a topology advertisement to be the *minimum* of all the levels of the channels it advertises.

Because we expect that realm boundaries will be administratively determined and configured—especially for boundaries at higher levels in the hierarchy—we assume that nodes know the levels of their attached channels, and include that information in each topology advertisement they send out.

Observe that, if no advertisement messages are lost, a node will see exactly two advertisements that mention each channel in its view of the topology—one emitted by each end of the channel. We say a node's view of the level-k topology has *converged* when it has seen two advertisements (including those it originated) of each channel with level at most k.

We are now in a position to state the simple rules governing origination and forwarding of topology advertisements in the presence of arbitrary realm nesting.

1) A node emits topology advertisements over links with levels greater than k only after its view of the level-k topology has converged.
2) The topology advertisement originated by a node over a channel with level=k contains exactly those channels with levels $\geq k$ reachable from that node via channels with levels less than k.
3) An incoming topology advertisement with level=k is forwarded over a channel with level=l if and only if $k \geq l$.

Together, these rules define the way topology information is aggregated by the border nodes of nested realms. The result is a set of hierarchical, location-specific abstractions of the network, along the lines of Alaettinoglu's *viewserver hierarchy* [2]. (In other research, we have shown how such a hierarchy can be built *without* node addresses [13].) An example is shown in Fig. 5. The advertisements originated over four of the links are shown schematically, with the advertising node/realm represented by a circle, and its border channels indicated by labeled edges; the level of each channel is shown next to it. The presence of a "T" in the message indicates that transit service between the channels is offered. The two messages in the bottom half of the figure would be emitted first, followed eventually by the upper two. Note that the advertisements sent over p and q contain the same information, as required.

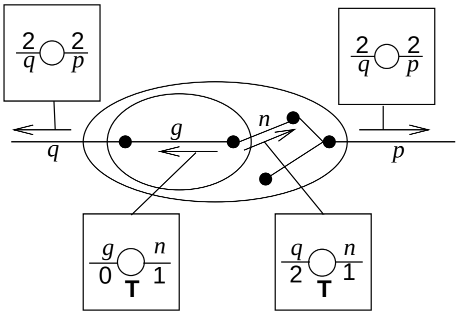


Fig. 5. Selected advertisements for the topology of Fig. 4.

D. Hierarchical Locators

Given an arbitrary hierarchy of realms, we also need to modify assumption P3. To that end, we introduce the notion of a *locator*, which generalizes the role of the border channel in P3, to indicate the location of the channel associated with a particular EID in the topology. More precisely, a locator encodes a set of ingress paths that can be used to reach a particular end-channel. Suppose, for concreteness in our discussion, that the destination endpoint is a server program, and there are three enclosing realms (corresponding to the *host*, an *OSPF area*, and the destination *Autonomous System (AS)*—refer to Fig. 6). A locator defines a set of sequences of ingress channels of the enclosing realms. The locator "$(A|B|C)$ then $(D|E)$ then $(F|G|H)$ then I" indicates that the endpoint resides in the (AS) realm identified by channel IDs A, B, and C, the inner (OSPF) realm identified by D and E, the host realm identified by F, G, and H, and finally the endpoint (EID) represented by channel ID I. Because locators only specify a partial path, final path selection is delayed until

[2] Although realms and nodes play the same role in our model—both terminate and provide transit between channels—we use the term "node" here to denote an atomic entity with no discernable internal structure, like a hardware FN.

packet forwarding time, and is made according to the policies of the destination realm(s).

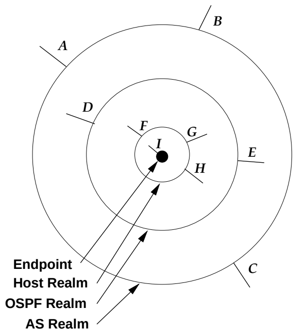


Fig. 6. An example locator specification: $(A|B|C)$ then $(D|E)$ then $(F|G|H)$ then I.

Locators are context-dependent, in the following sense: depending upon the location of a source, different resolutions of the locator will be appropriate. For a source located within the same (lowest-level) realm as the destination endpoint (e.g., the host realm), the EID is sufficient to determine a path to the endpoint. Within the same second-level realm (corresponding to, say, an OSPF subarea), the ingress channels of the lowest-level (host) realm and the endpoint are sufficient. A source in a different second-level realm but within the same top-level realm (corresponding to an Autonomous System) needs to know the endpoint's, first-level realm's, and second-level realm's ingress channels. Finally, a source in a different top-level realm needs the ingress channels of all three enclosing realms and the EID. In each case, the EID-to-locator service needs to return a locator with context-appropriate resolution.

Now we can modify P3 to deal with arbitrary levels of hierarchy:

P3a: There exists an EID-to-locator mapping service. Given the ID of any end-channel in the network, the service returns a locator for that EID, at the appropriate resolution for the requesting location. Every realm contains at least one provider of this service, and every node in the realm knows the identity of that provider.

We now describe a method by which an endpoint's locator is constructed, and how a locator can be obtained for a given endpoint. Each realm provides an EID lookup server for the purpose of constructing locators and mapping EIDs to locators. Each EID lookup server knows the path to its "parent" server (i.e. the server for the realm that contains it), and the path to each of its "child" servers, for the realms it contains. In larger realms, the service may be provided in a distributed fashion, e.g. through a DHT. Finally, there is a single global DHT EID lookup service at the top-most inter-realm level.

When an endpoint first connects to the network, it is given a forwarding directive and motivation token to contact the the local EID lookup server for the innermost-containing realm. The new endpoint informs that local server of its EID; the local server then sends the EID *together with the appropriate border channels of the local realm* to its parent server. The parent server repeats the process, adding the border channels of *its* realm, according to the policies of that realm. This continues until the top-level realm server is reached. The top-level server registers the complete locator with the EID in the global DHT.

To resolve an EID to a locator, a source first sends a resolution request to its local EID lookup server. If the local server has an entry for the EID, it responds with the (local) locator(e.g., the EID itself). Otherwise, the local server forwards the request to its parent server. This continues until the EID is found or the global DHT is reached. If the EID is found in the global DHT, the full locator is returned to the requestor. (Note that requests are forwarded along preconfigured or semipermanant paths among the infrastructure components, while responses follow the reverse paths.)

A slight modification of the above approach allows the particular entry channels returned in the locator to be dynamically adjusted by the destination realms, for example in response to changing traffic conditions. The cost is a slightly longer resolution path. Instead of storing the locator itself with the EID, the EID-to-locator server binds the EID in its parent server to a pointer to *itself*. The parent does the same thing, resulting in a downward chain of FDs from the top-level (global) service. Each EID-to-locator server keeps track of the conditions that determine the preferred ingress channels for its realm. A resolution request from outside the realm is forwarded down through the chain to the local realm server, which chooses the ingress channels and sends them back to its parent. Then, as the response follows the reverse path back up the hierarchy, each server adds the *current* "best" set of ingress links for its realm to the locator.

E. Dedicated Topology Infrastructure

So far we have assumed the participation of every node in topology discovery and route selection. In general, this is not necessary; it is sufficient for each node to know a path (forwarding directive) to a node that has the topology information specified in P1a and P2a. We therefore introduce a hierarchy of *topology servers*, which participate in the topology discovery algorithm on behalf of the nodes in each realm. The topology server periodically emits announcements of its existence, including the level of the realm for which it is responsible. These announcements are flooded throughout the realm like topology advertisements, accumulating a path (via the accountability field) as they go. In this way, each node learns a path to the servers for each of its enclosing realms, and lower-level servers learn paths to those further up in the hierarchy.

Each node (end system or FN) is still required to send out periodic topology advertisements; however, these are sent directly to the local topology server instead of being flooded throughout the enclosing realm. Each lower-level server then creates the aggregated advertisements for its realm and sends them to the appropriate higher-level server. This reduces the overhead of flooding, at the cost of having a dedicated server for the realm, and requiring an additional communication step (requesting topology information from the server) before sending a packet.

A similar arrangement applies to the *route selection* function. Users that do not need the ability to apply their own policies to select routes may delegate the task to a *route server*. They need only be configured with an FD that allows the to reach the route server.

V. Additional Considerations

This section briefly discusses some miscellaneous aspects of the approach.

A. Caching for Performance

Decomposing the network layer into a set of cooperating services (i.e., forwarding, topology discovery, path fault handlers, route selection, etc.) is key to the separation of mechanism from policy in our approach. However, this decomposition raises some issues with respect to performance. Although the goal is to support single-packet exchanges in a single round trip, the number of distinct, possibly distributed entities potentially involved in forwarding a single packet is a potential cause for concern. The obvious answer is that *caching* will play a crucial role in attaining adequate performance. We have already discussed caching in regard to path fault handler. In addition we expect that nodes will cache any information obtained from dedicated infrastructure such as the topology service, the EID-to-locatorservice, or any route service. Given that the information obtained from these services is likely to change slowly, we believe that caching will be quite effective in reducing overhead.

We plan to make use of *ephemeral state* [6] for caching wherever appropriate. Information stored in an ephemeral state cache persists for a relatively short, fixed duration before it automatically disappears. A cached entry cannot be deleted early nor can its lifetime be extended. As a result, the amount of state consumed by cached entries can be predicted and bounded. Moreover, there is no management overhead.

B. Multiplexing and Higher-Level Protocols

So far we have not mentioned layering or upper-level protocol processing, and have even implied that the endpoint itself is responsible for constructing the PFRI header and prepending it to the packet. This is because PFRI is designed to minimize assumptions about how systems and even protocols are structured, and to be strictly agnostic about the protocols being used above and below it. In practice, PFRI will be implemented in some combination of library and operating system modules. Fig. 7 shows one possible arrangement: the upper-layer protocols and per-endpoint portion of the PFRI implementation are accessed as libraries, while the rest of PFRI—viz., the part that multiplexes packets over the *access* channel x—is contained in the operating system. Because PFRI is agnostic about the nature of the entities in between channels, various other configurations are possible. For example, in a large "server farm", the last *several* links might be inside a cluster of machines.

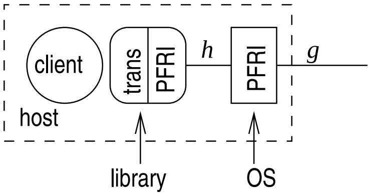


Fig. 7. Implementation structure

The question remains, however: what happens to a packet when it is received over the final channel in its FD? If it is an end-channel, the PFRI layer's work is done. Otherwise—an example would be the case of a gap-filling FD pushed onto the packet by a path fault handler—the packet must be processed by the receiving PFRI module. The PFRI header therefore contains a small type field that allows to determine which control or signaling function is or should be applied to the packet. Note that this field is used *only* to distinguish among network layer functions such as topology discovery, flooding infrastructure server announcements, path fault handling, etc. The PFRI header contains *no* information about any higher-level protocol used by the application; this is an intentional design decision, to prevent the infrastructure from basing its treatment of the packet on (implicit) assumptions about the application's needs, rather than explicit requests, provided via the motivation and knobs fields.

C. Bootstrapping: Joining the Network

To join the network, a node establishes an access channel to a forwarding node. It then needs to obtain the basic information required to enable it to communicate with the infrastructure. This information comprises at least the following: (i) a path to the local topology server; (ii) a path to the local EID-to-locatorservice; (iii) a path to one or more inter-realm route selection or brokering services; (iv) paths to any other higher-level resolution infrastructure (DNS analogues) as discussed in Section III. (For each of these paths, accompanying motivation credentials are also required.) Once this information is obtained, the node can obtain what it needs from the infrastructure to communicate with other endpoints. The simplest way to obtain this information is to get it from the forwarding node on the other end of the access connection. However, there are two reasons this may not be feasible. First, a FN—a dumb device that does nothing but forward packets—may not have some of it. Second, the service provider may wish to exercise control over which infrastructure components the joining node uses. Therefore the FN may be configured to provide a single path to a joining node, namely the path to a *boot server*. The boot server, in turn provides all the information needed by the joining node to bootstrap its communication.

VI. Related Work

As noted in the Introduction, the PFRI design draws on many sources. Here we trace a little bit of this heritage; a more comprehensive listing may be found in [4]. We note that what sets PFRI apart from many of its predecessors is the context in which it is intended to operate, that is, the inclusion of motivation, accountability and other mechanisms.

PFRI has much in common with the Nimrod routing architecture [7] in terms of both goals and methods. Nimrod aimed to support service-specific user-directed routing, mobility, and scalability. It used loose source routing based on hierarchical topology maps. The New Internet Routing Architecture (NIRA) [16] also includes mechanisms to support source selection of transit routes. In addition, packets carry information enabling providers to charge for transit service (like our motivation). PFRI differs from both of these in identifying channels instead of nodes, and in avoiding hierarchical identifiers altogether.

Recent work on Routing on Flat Labels (ROFL) [8] investigates the feasibilty of routing without the use of hierarchical, topology-based addresses, and concludes that it may not be a completely crazy idea. The approach presented here differs in several respects, including the resolution of an identifier to a location in the topology, and the emphasis on source routing.

The Forwarding and Control Element Separation (FORCES) working group of the IETF [1] defines protocols for decomposing the data plane and control plane functionalities of an IP router. That effort does not seem to have considered the problem of begging the question, as its protocols all run over IP.

VII. Conclusions

We have outlined a routing/forwarding architecture based on the use of channels as the primary abstraction, using a single, flat namespace. Our goals are to recognize that users and providers have different interests, and provide mechanisms that allow each party to express its own policies. We have identified the components of our architecture; a significant challenge is how to establish that our design really solves the problems it sets out to solve. The only way to really do this is to build it and use it on a daily basis. Even if it is superior, however, it may not "succeed" in overcoming the inertia of the existing Internet. Our hope is that, over time, we can learn enough to build a network that we ourselves prefer to use. Toward that end, our intent is to build a system that supports the "right" variety of options in terms of both mechanism and policy.

It is not entirely obvious that a high degree of flexibility should be a primary goal of the architecture, especially flexibility with respect to mechanisms. Experience has shown that even when a great deal of flexibility is provided, over time a small subset of choices dominates, so that ultimately the expense of providing the flexibility is often not worth the benefit [5]. The challenge is that at this stage of the design we cannot predict which techniques will be winners, so it is important to preserve options. By building, experimenting, and living with a flexible system, we can hope to learn enough to later go back and develop a more streamlined design. The challenge is to do this before the first version becomes too firmly entrenched. The canonical example of this problem is of course IPv4 and IPv6.

Many of the techniques used in our design are not new, having been proposed and even implemented in other contexts. However, the Postmodern Internet Architecture project is driven by the notion of starting with a clean slate while keeping in mind what has been learned in the past three decades. The hope is that ideas can combined and leveraged in a novel way to create an architecture that cleanly separates policy from mechanism, allowing tussles to be resolved outside the forwarding plane.

Acknowledgment

This work was supported by the U.S. National Science Foundation under grants CNS-0626918 and CNS-0435272. The authors are greatful to Bobby Bhattacharjee, Neil Spring, James Sterbenz, and Onur Ascigil for useful conversations. They should not be assumed to concur with anything in this paper. Errors, omissions, and silly notions are the responsibility of the first author.

References

[1] FORCES Working Group. http://www.ietf.org/html.charters/forces-charter.html.

[2] Cengiz Alaettinoglu and A. Udaya Shankar. The Viewserver Hierarchy for Interdomain Routing: Protocols and Evaluation. *IEEE Journal of Selected Areas in Communications*, 13(8):1396–1410, 1995.

[3] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proceedings of ACM SIGCOMM 2004, Portland, Oregon*.

[4] B. Bhattacharjee, K. Calvert, J. Griffioen, N. Spring, and J. P. G. Sterbenz. Postmodern internetwork architecture, 2006. Technical Report ITTC-FY2006-TR-45030-01, University of Kansas.

[5] K. Calvert. Reflections on Network Architecture: an Active Network Perspective. *ACM SIGCOMM Computer Communications Review*, 36(2):27–30, April 2006.

[6] K. Calvert, J. Griffioen, and S. Wen. Lightweight Networking Support for Scalable End-to-End Services. In *Proceedings ACM SIGCOMM 2002, Pittsburgh, USA*, pages 265–278, August 2002.

[7] I. Casteneyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. RFC 1992, August 1996.

[8] M. Cesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *Proceedings of ACM SIGCOMM 2006, Pisa, Italy*, pages 363–374, August 2006.

[9] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet (position paper). In *Proceedings of ACM SIGCOMM 2002, Pittsburgh, USA*, pages 347–356, August 2002.

[10] D. D. Clark. The design philosophy of the DARPA internet protocols. In *Proceedings ACM SIGCOMM 1988, Stanford, USA*, pages 106–114, August 1988.

[11] V. Jacobson. Compressing tcp/ip headers for low-speed serial links. RFC 1144, February 1990.

[12] J. Moy. OSPF Version 2. RFC 2328, April 1998.

[13] L. Poutievski, K. Calvert, and J. Griffioen. Link-state routing without addresses. in preparation.

[14] B. Raghavan and A. Snoeren. A System for Authenticated Policy-Compliant Routing. In *Proceedings of ACM SIGCOMM 2004, Portland, Oregon*.

[15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications, August 2001.

[16] X. Yang, D. Clark, and A. Berger. NIRA: A New Internet-Domain Routing Architecture, August 2007.

[17] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proceedings of ACM SIGCOMM 2005, Philadelphia, USA*, pages 241–252, August 2005.