Defining and enforcing transit policies in a future Internet

Jad Naous[†], Arun Seehra[‡], Michael Walfish[‡], David Mazières[†], Antonio Nicolosi[§], and Scott Shenker[¶]
[†]Stanford [‡]UT Austin [§]Stevens Institute of Technology [¶]UC Berkeley, ICSI

Abstract

Policy is now crucially important for network design: there are many stakeholders, each with requirements that a network *should* support. Among many examples, senders have an interest in the paths that their packets take, providers have analogous interests based on business relationships, and receivers want to shut off traffic from flooding senders. Unfortunately, it is not clear how to balance these considerations in principle or what mechanism could uphold a large union of them in practice. To bring the policy issues into focus, we (ironically) avoid predictions about which policy requirements will predominate in a future Internet and instead seek the most general policy framework we can possibly implement. To that end, this paper articulates a general policy principle; in condensed form, it is *empower all stakeholders*. Upholding this principle in the context of Internet realities, such as malicious participants, decentralized trust, and the need for high-speed forwarding, brings many technical challenges. As an existence proof that they can be surmounted, this paper describes the design, implementation in hardware, and evaluation of a concrete architecture.

1 Introduction

This paper is about the future of the Internet, but we begin with its past. The history of network routing began as a topological problem: how do you find the set of shortest paths in a network graph ([23])? However, with the advent of domain-based routing in the Internet, policy became an important consideration. In fact, policy concerns were embedded in the 1989 requirements document (RFC 1126) that set the groundwork for the first version of BGP:

> Those resources used by (and available for) routing are to be allowed autonomous control by those administrative entities which own or operate them. Specifically, each controlling administration should be allowed to establish and maintain policies regarding the use of a given routing resource. [43]

Embodying this principle, BGP allows each domain to decide, unilaterally, which routes it accepts and exports based on the full AS-level path.

Provider control is no longer limited to the control plane; providers have imposed usage limits and blocked certain types of traffic that they believe would be injurious to their or other networks.

Moreover, ASes are not the only stakeholders in the Internet. There have been many calls to grant sources some control over their packets' paths (see, for example, [25, 28, 33, 39, 52, 53, 55, 65, 66]). The reasons for source control vary from performance (letting sources find the best quality paths) to preference (letting sources avoid providers they don't trust) to price (letting sources find the cheapest paths).

For exactly the same reasons, receivers, too, have an interest in controlling the path of their incoming packets. Receivers also care *who* is sending them packets and may wish to allow only a subset of incoming flows (e.g., when under attack, accept packets only from their known customers).

While all of the participants could legitimately be considered stakeholders, and the policy concerns discussed above seem plausible, it is not clear how to balance the various concerns. Take, for instance, the case of a user trying to send email from her hotel room. The user would like her packets to reach her company's mail server via a reliable and high-bandwidth path. The hotel would like her packets to take the least costly path. The first-hop provider cares that the packets are coming from a paying customer but wants to block all transiting SMTP traffic because they fear it might be spam. The receiving mail server only wants to receive outgoing SMTP traffic from company employees. Moreover, it wants all of this traffic to pass through a third-party virus-scanner service to which it has subscribed. All of these are valid policy goals, as they concern the use of the stakeholder's resource or the fate of their own communication, but it is not clear which of these policy considerations, when they are in conflict, should prevail.

All of the preceding background leads us to the question this paper tries to address: what policy framework should we adopt in a future Internet architecture? This question is one of both policy and mechanism: what policy considerations should the architecture support, and can we build a mechanism to support those considerations?

Judging by the increasing number of architectural proposals that support policy-oriented features such as interdomain policies, source selection of routes, and interposition of middleboxes by endpoints, there appears to be consensus that the various stakeholders have the right to exert some control over their flows, and that these considerations should be reflected in a future Internet architecture. Table 1 lists many, but by no means all, of these proposals. As the table shows, the union of policy considerations is large, but the intersection is small: each proposal generally supports only a particular subset of stakeholder control, and many of these works don't compose.

Thus, as a community striving to design the future Internet, we have two choices:

- Choose one subset of policy considerations and bet that it will be sufficient to meet all policy needs for the foreseeable future.
- Develop a flexible architecture to support all reasonable

Approach	dest. can constrain sender	resource attribution	provider policy granularity: prefix	provider policy granularity: suffix	provider policy granularity: subsequence	src can constrain routes	MB* can constrain routes	rcvrs can invoke MBs*	providers can invoke MBs*
BGP				x				* MB = middlebox	
Capabilities [63, 67]	x								
Filters [11, 19, 24, 34, 36, 45, 46, 61, 64]	x								
Intserv, RSVP [13, 14]	x	x							
Visas [26]		x							
Platypus [55]		x				x			
LSRR [6]	x					x			
Policy routing, Nimrod [18, 21]					x	x			
Pathlets [28]				x	x	x			
Wiser [47]			x						
MIRO [62]				x					
RBF [54]	x					x		x	x
Src routing [25, 33, 39, 65, 66]						x			
Byzantine routing [52, 53]						x			
NUTSS [32]							x		
i3, DOA [58, 60]								x	
DONA [41]									x
IGLOO (this paper)	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**

Table 1—Policy controls available in many, but not all, network-layer proposals. Abstracting the details of the various proposals, each of the listed controls (columns) can be viewed as some entity along the path of a communication constraining some portion of the path of the communication. While many of the listed proposals (rows) cannot be implemented together, the framework in the text is intended to make available all of the controls in the columns, and many more besides. As argued in the text, each of these controls represents a legitimate policy interest on the part of some stakeholder.

policy considerations, allowing the Internet's policies to evolve as its usage and organizational structure change.

The first choice, while certainly expedient, seems risky given how unpredictable the Internet has been so far, in both the nature of its traffic and the organizational structure of its stakeholders.[1] In fact, we (as a community) have a terrible record in predicting the future of the Internet, and opting for this choice is a gamble that we will finally get it right this time.

Thus, on policy grounds, the second choice is more desirable: it would not only accommodate the many stakeholders and desirable policy goals that have been proposed but also avoid guesses today about what will be important tomorrow. However, it poses two challenges: can we identify what constitutes *reasonable* policy considerations, and can we build a mechanism to support all such policies? In response to the first challenge, we offer the following principle:

Policy principle: *A communication should be allowed if, and only if, all participants approve. By participants, we mean the sender, the receiver, the carriers, and any other intermediaries.*

In the next section, we elaborate on the principle and describe the controls that it implies. Here, we just note that the principle—though it may seem like overkill because it apparently gives every participant veto power—in fact appears necessary to support all potential reasonable desires of all stakeholders. Moreover, these policy concerns are real: senders and receivers (at least some of them) do care about which ASes their packets transit, receivers do want to block traffic from troublesome senders, and all participants do have business and security reasons to favor particular parts of the network, or specific middleboxes.

[1] Recall that the modern ISP-oriented Internet arose in the last fifteen years and is not at all what the Internet pioneers envisioned.

Of course, we cannot know whether the principle will ultimately be *sufficient*, but, encouragingly, the controls it implies appear to encompass all prior transit policy proposals.[2] Thus, it at least provides a useful minimum target for any architecture seeking to support all potential reasonable policy considerations of all stakeholders.

What about the second challenge: can we build an architecture that supports such a general set of policies? To answer this, we ask a few more questions:

Can we first design such an architecture? Yes. This paper presents IGLOO, a new network architecture that appears general enough to enforce the policies of all prior transit policy proposals. Like prior works [3, 15, 17, 30, 31], IGLOO separates the control plane from the data plane: policy decisions in IGLOO are expressed on commodity servers, vastly lowering the barrier to realizing and enforcing new policies. However, unlike those works, which are designed for enterprise networks, IGLOO targets a federated network like the Internet that has no global root of trust.

Federated networks bring technical challenges. For example, because participants may want to enforce policies based on a communication's inter-domain path, IGLOO's data plane must solve a longstanding problem in network architecture: verifying that packets are actually following their approved paths. As summarized in Table 2, previous solutions require a central authority, limiting their scope to single domains. IGLOO's solution to this problem introduces packet authentication techniques that may be of independent interest.

[2] By transit policy we mean any policy that is based on the nature of the flow rather than on individual packet contents. Our policy space cannot accommodate policies like drop all virus-laden packets, but it can support policies that require flows to pass through a virus-scanning middlebox.

Mechanism	all participants can deny based on path	comm. held to described path	malicious behavior tolerated	decentralized	fixed and feasible data plane
IP+BGP (the status quo)				x	x
Ethane		x	x		x
Auditing [68]			x		x
MPLS, virtual circuits, resource reservation [10, 14, 56]	x			x	x
Capabilities, Platypus [55, 63, 67]			x	x	x
Passport [44]			x	x	x
Byzantine routing [52, 53]		x	x		
Secure routing [8, 50]		x	x	see caption	x
Secure policy routing [27]	x		x	see caption	
PoMo Architecture [16]	x	x	x		
IGLOO (this paper)	**x**	**x**	**x**	**x**	**x**

Table 2—Prior approaches to aligning control and data planes, in terms of our requirements. One reason that MPLS, for example, does not tolerate maliciousness is that two entities on the intended path can collude to skip a third; more generally, MPLS doesn't include cryptographic assurance to provide proof that a packet is following its approved path. Secure routing and secure policy routing do not require a PKI, but they do require prior coordination and pre-configuration among the hops; thus, they don't fully meet our decentralized requirement.

Is the architecture feasible to implement? Section 6 describes a working IGLOO prototype, built on NetFPGA [2].

Can we avoid the architecture having to pay for unused generality? Yes in the control plane, no in the data plane. Though IGLOO's control/data plane split allows any entity on a communication's path to implement arbitrarily complex policies, participants with simple policies may want to disintermediate themselves from the approval process. For instance, a provider may wish to approve any traffic destined to a customer so long as the customer also approves it. In this case, the provider does not want to be consulted on each of the customer's communications. IGLOO lets such a provider entirely delegate this slice of policy to the customer's control plane. However, the provider's data plane must still enforce the customer's policy to avoid carrying unwanted traffic. In effect, a corollary of moving policy to the control plane is that simplifying policy does not simplify the data plane.

What is the cost of moving to such an architecture? Section 7 reports that our prototype forwards at rates comparable to IP, but using 86% more logic area than an IP router and an average packet space overhead of 23%. These costs are expensive today (though not prohibitive). However, we are designing for the future, and technology trends often make today's expensive design tomorrow's commodity.

Can the architecture scale to the Internet? Plausibly. We obviously have not tested at scale, but the data plane looks likely to scale (Section 7.4) and a working control plane (Section 5) is designed to handle inter-domain routing and authorization at Internet scale. We would be surprised to see this specific control plane deployed Internet-wide without substantial enhancements. Still, the working control plane is evidence that Internet-scale deployment of some IGLOO control plane is plausible. Moreover, as noted in Section 5, IGLOO's data plane can simultaneously support a wide range of conceivable control planes, including ones that mimic the policies of prior proposals (BGP, Pathlets [28], NIRA [65], etc.), with the bonus that such policies can be *enforced* (today, such policies are only advisory [48]).

The contributions of this paper are the policy principle and the answers to the above questions.

But there is a higher-level point to this paper. In the near-term, network architecture research does not seek to settle debates but rather to explore the space of the possible. IGLOO demonstrates that upholding the policy principle is possible. Indeed, while we cannot answer whether the benefits will outweigh the costs (particularly as technology evolves to diminish those costs), we can now at least *ask* what we would have to pay for the generality. Of course, our notion of abundant generality may later be seen as insufficient, which would call for a broader policy principle. But even in this case, we hope IGLOO's ideas can serve as a stepping stone to an even more general network architecture.

2 What are reasonable policy considerations?

This section elaborates on the policy principle. The sections ahead describe how IGLOO enforces policy.

The policy principle posits that every entity along the path of a given communication (end-hosts, providers, and other intermediaries) must approve the communication or else the network should deny that communication. Moreover, *only* those entities along a communication's path should be able to deny it. The principle raises three questions:

1. Why should the power to deny be vested *only* in entities along the path of a communication?
2. Why should the power to deny be vested in *every* entity along the path of a communication?
3. What are *reasonable* inputs to these deny decisions, given that we cannot predict the future of the Internet?

Question 1 asks why non-participants in a given communication get no say in that communication. Our answer is that while the policies of such third-parties (governments, etc.) might be highly relevant, they should be addressed by non-network means, such as the legal system. Why? Our position is that the Internet architecture should empower the Internet participants. If there are actors whose authority to control communication derives from a source besides participation, such as laws, then they should use that authority to restrict the policies established by participants. For example, if the US Congress wants to enforce network neutrality, it should pass laws stipulating that certain participants *cannot*

deny communication. We strongly believe that one should not embed such legally derived concerns into the network architecture itself, as they are likely to differ from region to region and change over time.

To Question 2, as noted in the introduction, any entity along the path of a communication, including the two endpoints, is a legitimate stakeholder: its resources are used for that communication, and it is likely to care a great deal about whether and how those resources are disposed.

One concern is that this view seems to explicitly empower all entities along the path of a communication to drop the communication (versus the implicit power they have from being located in the path). But note that just because the Internet architecture makes a control available does not mean that it will be exercised, as economic and social pressures strongly constrain which policies are enacted. For instance, under BGP today, autonomous systems can pick routes based on the entire interdomain path, but they rarely exercise more than first-hop preferences. And in the future, perhaps paths will be chosen by edge providers, with carriers caring only whether they can bill a principal [55].

Our point with these examples is that, as noted in the introduction, we think it unwise to embed policy predictions in a long-lived architecture. Instead, our approach is to strive for an architecture that upholds all reasonable policy considerations, letting the future decide which controls get "lit up".

This brings us to the third question: what are *reasonable* inputs to an entity's decision to approve or deny a communication, and how can we design for future policy considerations if we do not know what inputs will be required?

We can never answer such a question definitively. However, it helps to look at previous attempts to enforce policy by enhancing the network architecture. After all, if our definition of reasonable cannot encompass the goals of the present, it has no hope of addressing the future. Table 1 lists some 30 prior works; in each case, one or more participants can establish policies based on the identity (or behavior) of some of the other participants. As examples, in capability or default-off architectures [11, 24, 45, 63, 67], the receiver elects not to hear from particular senders; in NIRA [65], the source controls a prefix of the inter-domain route, and the receiver controls the remaining suffix; in BGP, providers constrain the prior hop and make decisions based on the downstream AS path. One can also imagine a provider caring about who else has carried or will carry a packet, say to uphold promises of particular classes of service to either endpoint.

Generalizing slightly, any participant in a communication may reasonably want to approve or deny communications based on the identities of all other participants. In other words, *the sender, full inter-domain route, and receiver, which we call the* path*, should be an input to each participant's decision function.* While any given participant will likely care only about specific parts of this input, to not provide it in its comprehensive form would potentially rule out reasonable (perhaps unanticipated) policy considerations.

But is the path enough? A participant also cares which of its local resources a communication consumes. Is the communication long- or short-haul traffic? Is it transiting the participant's network or terminating there? What QoS properties does the traffic demand? How should the traffic be shaped?

Participants may have a slew of other considerations, including user authentication, billing status, time of day, traffic type, packet contents, and whether the *other* participants agree to the communication. For example, in keeping with the policy principle, if an intermediate provider knows that the destination does not want to receive a given flow, that provider ought to decline to be part of that flow.

Our IGLOO network architecture, which we turn to in a moment, allows entities to express and enforce the above policy considerations. Specifically, an entity gets the opportunity (which it might not exercise) to approve flows based on who is participating (the path), what *local* resources the communication will consume (including links, queuing priority, required middleboxes, ports on end-hosts, paths through an internal network), and arbitrary other inputs (billing status, time of day, etc.). Then, the network ensures that flows obey these constraints.

Note that under IGLOO, entities cannot enforce policies about what other entities do internally. One reason is that providers need the freedom to innovate and evolve their networks transparently. Thus, an entity under IGLOO cannot enforce *how* the other participants handle a communication—including whether a provider used a particular lambda, whether it delegated its role to another provider in another country, where a packet traveled geographically, whether a packet was copied, etc. Customers must enforce such concerns outside the architecture (e.g., through negotiated contracts). However, IGLOO at least exposes a foundation for the needed negotiations. This contrasts with the status quo, in which participants have little control over the characteristics of the flows they originate, carry, and receive.

We now delve into IGLOO's details, describing how it enforces the policy considerations just described.

3 Data plane

IGLOO's data plane is responsible for forwarding data packets and enforcing policies established by the control plane. This section describes both the interface through which one expresses policy to the data plane and how the data plane's dedicated forwarding hardware processes packets.

The data plane is intended to provide a fixed target for hardware manufacturers while permitting a range of possible control planes. The control plane resides off the data path in software on commodity servers, making it easier to extend so as to support new policy considerations. Many policy-dependent functions traditionally handled on the data path are thus implemented by IGLOO's control plane, not its data plane. For example, a packet's full inter-domain path is determined before it is first transmitted, not while it is en route.

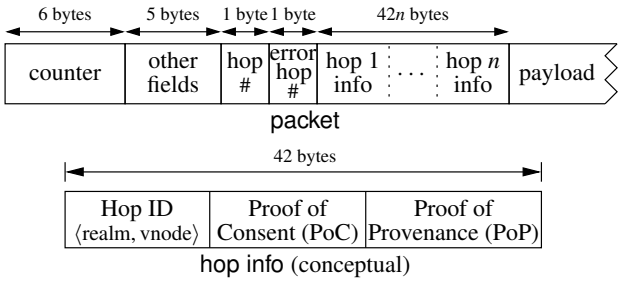


Figure 1—Logical packet format. The *hop info* depicted is conceptual; Section 3.6 describes how the actual hop info is compressed to 42 bytes. (§7.1 and §8 discuss whether 42 bytes is still too high.)

In general, a machine needs to communicate with control plane servers before it can use the data plane. There are two ways in which such communication takes place: over the link layer of the local network—using a broadcast host-configuration protocol analogous to DHCP—and over IGLOO's own data plane. The former method is used to bootstrap the latter. Bootstrapping is a special form of delegation, which we discuss in Section 3.3. For now, we simply assume hosts can invoke the control plane when necessary.

3.1 Overview

The data plane enforces policy by dropping packets that are not part of an authorized communication. Enforcement boils down to three requirements: a forwarder must verify that a communication has been *authorized* by the control plane; *authenticate* a packet that claims to be part of an authorized communication; and *constrain* a packet's intra-domain route, queuing priority, or other consumption of local resources as stipulated by the control plane. The three requirements are reflected in IGLOO's packet format, a simplified version of which is shown in Figure 1.

IGLOO divides the network into autonomous *realms*, each of which is controlled by a single administrative entity. Realms are the participants in communications and thus the entities with policy desires. The forwarders in a particular realm enforce the policy established by that realm's control plane. (Cross-realm policies, such as upstream realms dropping traffic not wanted by downstream realms, are the purview of the control plane, and are discussed in Section 5.) Each packet contains *hop information* indicating all realms along its intended path (source, destination, and intermediary realms). Realms are analogous to BGP ASes, but, because of IGLOO's greater flexibility, can usefully be smaller. We envision realm granularity ranging between that of today's ASes and that of DNS zones.

Realms can authorize communications based on arbitrary criteria, from traditional routing policy considerations to credit card payments to reputation systems, but IGLOO shields the data plane from these details. The data plane only needs to know *whether* the control plane has authorized a communication, not *why* or *how*. Control plane servers could directly transmit approval of each new communication to forwarders, but we rejected this design because the state kept by forwarders would complicate fail-over. Instead, control plane servers signal approval by giving senders *Proofs of Consent* (*PoCs*), cryptographic values reminiscent of capabilities [55, 67]. Senders embed PoCs in packets, making IGLOO traffic *self-authenticating*.

When a communication is approved, the sender gains the ability to send packets along a path for some period of time. However, the data plane must still *authenticate* packets to ensure they are fully following the authorized path. Otherwise, valid PoCs could be misused by unauthorized communications (for instance to impersonate an authorized sender along a suffix of an authorized path). To prevent such abuse, packets additionally contain cryptographic *Proofs of Provenance* or *PoPs*, through which each realm handling a packet proves to subsequent realms that it has approved and authenticated the specific packet.

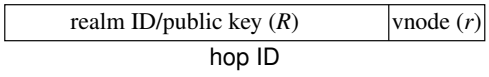
Finally, the path additionally contains a per-realm value called a *vnode* [28] that constrains the use of that realm's resources. The interpretation of vnode values is entirely up to the local realm, somewhat like MPLS labels. Vnodes can be used to restrict connectivity (e.g., deny long-haul transit), specify quality of service, or force traffic through middleboxes. To prevent unauthorized use of vnodes, PoCs simultaneously authorize both the realms and vnodes in a path. We elaborate on vnodes at the end of Section 3.2.

IGLOO is *decentralized* so as to scale to a federated network like the Internet. Realms need not trust each other except as required to carry traffic over a specific path. Moreover, the architecture does not require any central authority, not even a registry for realms. (We do, however, anticipate the need for at least one hierarchical naming scheme like DNS on top of IGLOO.) As summarized in Table 2, IGLOO is the first feasible network architecture to provide secure enforcement of paths without resorting to centralized trust.

The remainder of this section expands on the packet format and packet handling of the data plane.

3.2 Hop IDs

A packet's path consists of a list of *hop IDs*, each of which has two parts: a *realm ID*, which is a public key, usually designated R, and a *vnode*, usually designated r. Using public keys to identify network entities is by now a well-established technique [5, 49]. The vnode abstraction was introduced more recently by Pathlet routing [28].



The *realm ID*, R, names a realm participating in the communication, much like an AS number in a BGP AS path. Unlike AS numbers, however, which are centrally allocated, realm IDs have an egalitarian namespace because they are public keys; anyone can unilaterally create a new, globally-meaningful realm ID by generating a fresh key pair. This is crucial in light of our need to avoid centralized trust.

A forwarder in realm R learns of the existence of a remote realm, R', the first time it processes a packet whose path contains R'. If R' precedes R on the path, then the packet should already have transited R'. In this case, to authenticate the packet, the forwarder must check that R' approved it; this is done by verifying a cryptographic value in the PoP. Conversely, if R precedes R' on the path, i.e., the packet should subsequently transit R', R's forwarder must prove to R''s forwarder that R approved the packet; this is done by placing a cryptographic value in the PoP. The specific PoP computations are described in Section 3.4, but either case requires a shared cryptographic key between R's forwarder and R''s.

Realm IDs provide these shared keys, which we term *PoP keys*. Every forwarder in realm R_i knows the realm's corresponding private key, R_i^{-1}. Given another realm ID, R_j, the forwarder uses non-interactive Diffie-Hellman key exchange to compute two PoP keys, $k_{i,j}$ and $k_{j,i}$. R_i's forwarder uses $k_{i,j}$ (resp. $k_{j,i}$) to compute PoPs for (resp. verify PoPs from) R_j.

The use of public keys raises the question of key management. How do you know you have the right realm ID for what you want to do? This is analogous to asking how you know you are talking to the right computer. The Internet has shown that the answer depends highly on the specific use case. Often people don't actually care if they are talking to the right computer. Other times they rely on a central certificate authority (as SSL/TLS does), cache public keys (as SSH does), or manually distribute symmetric keys (as with TCP MD5 and sometimes IPsec). It would be a mistake not to support all reasonable key management schemes [49]. IGLOO therefore leaves key management to the control plane and the application, where, like today, multiple schemes can be introduced and happily coexist on the same underlying network.

Public keys further raise the question of key revocation. A realm's public key is also its realm ID; should the key or a derived PoP key be compromised, the realm will have to change IDs. This means reconfiguring forwarders (in itself relatively straightforward). In the control plane, it is tantamount to changing an organization's AS number—all routes through the old realm will be withdrawn and equivalent ones advertised by a new realm. To minimize disruption, forwarders can operate both realms simultaneously during a "make-before-break" transition period.

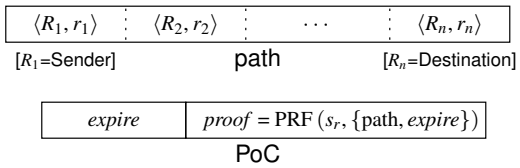
The *vnode* in a hop ID is a hook with which to bind approved communications to policy-dependent data-plane functions. Vnodes can affect queuing priority, restrict intra-domain routing, mandate middleboxes or traffic shaping, or be used to trigger unanticipated future data plane features.

Vnodes are also the granularity at which realms perform accounting and can delegate and revoke path authorization privileges. We envisage that providers will assign different vnodes to different customers and peers and, through delegation, allow these parties to authorize their own traffic over its network. Vnode-based restrictions on intra-domain routing can enforce a wide variety of policies [28] such as valley-free routing even if delegation allows customers to authorize arbitrary inter-domain paths.

3.3 PoCs

The purpose of a PoC is to prove to a realm's data plane forwarders that its control plane servers have authorized the particular path and vnode of a packet. Forwarders drop packets that do not contain valid PoCs. The authorization is proved by means of a *vnode key*—a symmetric key, s_r, specific to each vnode, r, and shared between all control plane servers and data plane forwarders in a realm.

Each PoC C has two parts: $C.expire$, the expiry time stamp, and $C.proof$, a token computed from the vnode key s_r via a pseudo-random function (PRF)—a sort of message authentication code (MAC) with deterministic outputs:

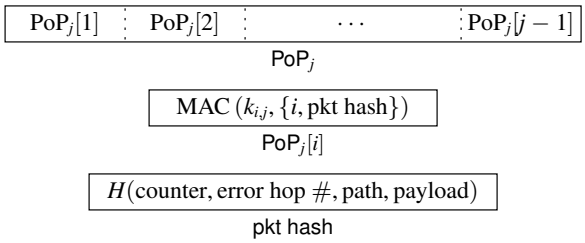


To delegate path authorization for a vnode, a realm distributes the vnode's key. A delegate uses this key to authorize arbitrary paths through the realm's vnode without further involving the realm's control plane servers. An important use of delegation is bootstrapping; IGLOO's host configuration protocol returns vnode keys with which hosts can talk to local control plane servers from which to obtain more PoCs.

A PoC states that a realm has authorized one of its vnodes to appear in a particular path. This does not always mean that connectivity actually exists along the path. If two back-to-back realms in the path do not peer, the PoC will be of little use for sending traffic. More interestingly, even when connectivity between two realms R_1 and R_2 exists, R_1 may choose to configure only some of its vnodes to forward traffic to R_2. This is how providers can enforce valley-free routing even after distributing vnode keys. E.g., if R_1 offers peering but not transit to R_2, then R_1 should give R_2 a vnode that only connects to R_1's customers, not its upstream providers.

3.4 PoPs

The Proof of Provenance, or PoP, lets a realm verify that a packet with a valid PoC is actually following the path authorized by that PoC. Forwarders drop packets that do not contain valid PoPs. Suppose a path consists of hop IDs $\langle R_1, r_1\rangle, \langle R_2, r_2\rangle, \ldots, \langle R_n, r_n\rangle$. When R_j receives a packet with this path, for the packet to be valid, its jth hop info (Figure 1) must contain a proof of provenance PoP_j. PoP_j consists of assertions by each realm R_i for $i < j$ that R_i has approved and authenticated the packet.



PoPs are computed using the shared PoP keys discussed in Section 3.2. Let $k_{i,j}$ be the PoP key for messages from R_i to R_j. Before R_i forwards a packet, it adds a cryptographic verifier to the PoP in every hop info from i onwards. Specifically, for each hop number j (where $j \geq i$), the forwarder in realm R_i uses $k_{i,j}$ to compute a MAC over the current hop # (namely i) and a collision-resistant hash of most other packet fields (except the PoPs, which are changing). It then appends these MAC values as $\text{PoP}_i[i], \text{PoP}_{i+1}[i], ..., \text{PoP}_n[i]$ before forwarding the packet.

Note that R_i adds a MAC for itself under key $k_{i,i}$ to PoP_i. (This is not shown in the diagram above, which depicts the state of PoP_j when a packet reaches realm R_j.) This $\text{PoP}_i[i]$ value lets R_i later verify that it actually forwarded the packet, which in turn helps ensure the legitimacy of error packets as described in Section 3.5.

All of the symmetric-key cryptographic algorithms were chosen so that our forwarding hardware could process packets at line rate. Unfortunately, the public-key algorithm used to compute PoP keys takes about 4 msec in our implementation (§7.3), far too slow to be done for every packet. The PoP key derivation cost is mitigated by the fact that forwarders aggressively cache PoP keys; a forwarder computes PoP keys only the first time a new realm appears in any path. IGLOO also gains a factor of two because forwarders do not derive $k_{i,j}$ and $k_{j,i}$ separately; they are actually the same, with MAC inputs tagged to avoid ambiguity.

3.5 Packet processing and errors

When a realm receives a packet, it does the following:

1. Verify the PoC, and drop the packet if the checks fail.
2. Use a replay cache to check if the ⟨PoC, counter⟩ pair is an obvious replay, and if so drop the packet.
3. Verify the PoP, and drop the packet if the checks fail.
4. Use the current hop's vnode and next hop's hop ID to determine an intra-domain route and output port.
5. Add MACs to the current and subsequent PoPs.
6. Increment the packet's hop #.
7. Forward the packet out the appropriate port.

Unfortunately, step 4 can fail, particularly following network topology changes—perhaps the current realm can no longer reach the next hop, or no remaining intra-domain routes for the next hop are permitted for the current vnode. Because packets are source routed, such a failure must be propagated back to the sender for it to select another route. (The sender may already have an alternate route from the original path negotiation, or may need to invoke the control plane again.) Requiring all route adjustments to be made at the sender potentially adds latency to failure recovery. On the other hand, BGP can take tens of seconds to converge [42], so the ability for senders to try new routes unilaterally can in some cases allow faster recovery.

To allow error reporting, whenever a realm consents to a forwarding path, it may (and usually does) implicitly consent to carry error packets in the opposite direction. To create an error packet, a forwarder sets the *error hop #* field in the header to the current hop #. (The error hop # field is 0 in regular packets.) An error packet's payload contains the original packet's *pkt hash* (§3.4), followed by optional error-specific information (analogous to ICMP error type/code and data).

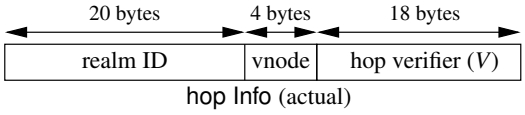
Forwarders recognize packets with non-zero error hop # fields as error packets and handle them differently. The most obvious difference is that these packets are routed backwards, to the previous hop, and that the hop # field must be decremented rather than incremented. A forwarder that experiences a failure when sending an error packet drops it at once—error packets never generate further error packets.

The PoPs in an error packet contain all the forward-direction MACs from the original packet that caused the error in addition to MACs for the error packet itself. Because the error payload begins with the original packet's pkt hash, forwarders can verify these forward-direction MACs despite not having the original packet. In particular, the forwarder in realm R_i drops an error packet unless PoP_i has a MAC under $k_{i,i}$. This ensures that a realm will not forward an error packet if it did not previously forward the original packet.

3.6 Design details

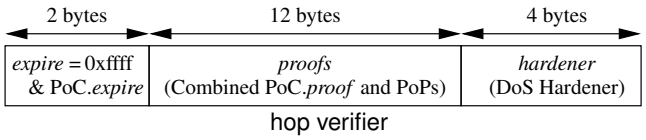
Here we describe the design in more detail, along with how the hop info is squeezed to 42 bytes.

Hop info The actual hop info used is shown below:



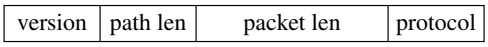
To make public keys small, we use elliptic curve cryptography (ECC). Every realm ID, R_i, is a point on NIST's B-163 binary-field elliptic curve group [4]; this affords roughly 80-bit security, similar to 1,024-bit RSA keys [4]. We reduce the representation of R_i from 163 to 160 bits by requiring the top three bits to equal a hash of the lower 160; the cost is a factor of 8 in expected key generation time. The hop ID, consisting of the realm ID and vnode, consumes a total of 24 bytes.

The remaining 18 bytes of the hop info hold a *hop verifier*—a combined proof of consent and provenance that lumps the hop's PoC and PoP values together as shown:



Each hop verifier V has three parts: a consent time stamp, $V.expire$, a 12-byte proof, $V.proofs$, and a 4-byte denial-of-service hardener, $V.hardener$. $V.expire$ contains the low-order 16 bits of the PoC's expiration time. (PoCs have a maximum lifetime, ensuring this value does not wrap.) $V.proofs$ is a 12-byte aggregate MAC [38] combining the realm's PoPs with a cryptographic derivative of PoC.*proof*.

Other packet fields Other packet fields are shown below:



Vnode keys and controlled delegation The vnode key s_r is used to derive PoCs for paths containing vnode r as follows:

$$\text{PoC}_i.proof = PMAC(s_r, \text{path} \parallel \text{PoC}_i.expire)$$

PMAC [12] is an efficient, parallelizable MAC and PRF.

It would be cumbersome for a realm to manage keys for its 2^{32} vnodes separately. Instead, we rely on a small number of shared *prefix keys* to pseudo-randomly generate many vnode keys. Let r/p denote the p-bit prefix of vnode r. If r/p is not explicitly bound to a shared prefix key and $r/(p-1)$ has prefix key $m_{r/(p-1)}$, then the prefix key for r/p is computed as $m_{r/p} = \text{PRF}(m_{r/(p-1)}, r/p)$ (a generalization of a technique suggested by [55]). The vnode key s_r is just $m_{r/32}$.

Given this approach to vnode key derivation, *vnode prefix delegation* is easy to implement. To delegate a block of vnodes with prefix r/p (i.e., 2^p vnodes) to a customer, the realm gives the customer a single prefix key, $m_{r/p}$. The customer can further sub-delegate by divulging $m_{r/k}$, where $k > p$. Additional optimizations are discussed in [51].

Packet construction and rewriting To initiate a flow, the sender invokes the control plane to retrieve PoCs (§5). To construct individual packets, it computes the initial values of the per-realm hop verifiers, $V_1, \ldots, V_n$, where n is the path length. Then, as the packet travels, each realm R_i follows the steps in Section 3.5, except for the following two changes:

- The realm combines steps 1 and 3 to verify both consent and provenance using V_i.
- The realm proves provenance (step 5) by modifying $V_i, \ldots, V_n$ (rather than increasing the length of PoPs).

We now describe the above steps in more detail. Our description uses the following functions. PRF-96 is a keyed pseudorandom function that maps 256-bit quantities to 128-bit quantities using two AES-128 encryptions, then discards the last 4 bytes to produce a 96-bit result. PRF-32 does the same but returns only the last 4 bytes of the 128-bit quantity. H-247 is a 247-bit suffix of the collision-resistant hash function CHI [35], a SHA-3 candidate. (Details about these functions are in our technical report [51].)

The sender initializes V_i for $1 \le i \le n$, as follows:

1. Set $H = H\text{-}247(counter \parallel err\ hop\ \# \parallel path \parallel others \parallel payload)$
2. Set $V_i.proofs = PRF\text{-}96(\text{PoC}_i.proof, 1 \parallel H)$
3. Set $V_i.hardener = PRF\text{-}32(\text{PoC}_i.proof, 1 \parallel H)$
4. Get $k_{1,i}$ from cache or by slow path calculation
5. Set $V_i.proofs = V_i.proofs \oplus PRF\text{-}96(k_{1,i}, 1 \parallel H)$

To verify consent and provenance, a forwarder in realm R_i, $1 < i \le n$, checks the PoC's expiry, reconstructs a reference hop verifier V' and compares it against V_i, the hop verifier in the ith hop ID in the packet, as follows:

1. Check $V_i.expire$ and drop packet if PoC expired.
2. Set $H = H\text{-}247(counter \parallel err\ hop\ \# \parallel path \parallel others \parallel payload)$
3. Set $V'.proofs = PRF\text{-}96(\text{PoC}_i.proof, 1 \parallel H)$
4. Set $V'.hardener = PRF\text{-}32(\text{PoC}_i.proof, 1 \parallel H)$
5. Check that $V'.hardener = V_i.hardener$ (and drop if not)
6. For $1 \le j < i$:
   (a) Get $k_{j,i}$ from cache or by slow path calculation
   (b) Set $V'.proofs = V'.proofs \oplus PRF\text{-}96(k_{j,i}, j \parallel H)$
7. Check that $V'.proofs = V_i.proofs$ (and drop if not)

After the above steps, the forwarder proves provenance by rewriting the hop verifiers as follows. For $i \le j \le n$:

1. Get $k_{i,j}$ from cache or by slow path calculation
2. Set $V_j.proofs = V_j.proofs \oplus PRF\text{-}96(k_{i,j}, i \parallel H)$

Note that the above steps might require the forwarder to derive some $k_{i,j}$ or $k_{j,i}$. However, the forwarder would expend such computations only *after* having verified $V_i.hardener$. Without this check, an attacker could invent realms and bogus paths to force spurious slow path operations on forwarders. $V_i.hardener$ is only 32 bits, so it does not rule out such attacks altogether, but it decreases their effectiveness by a factor of 2^{32}, which is sufficient to avoid denial-of-service.

4 Security considerations

As discussed in Section 3.1, IGLOO guarantees that the data plane will drop a packet unless the current realm has approved the path and all previous realms on the path have approved the packet. Based on the strength of the cryptography, these guarantees should hold even in the presence of malicious parties, so long as the local realm's control plane and forwarders have not been compromised. Nonetheless, even with these guarantees, malicious actors can still cause undesirable things to happen in the network. In some cases IGLOO can help mitigate these problems through the control plane. In other cases, the problems are future work, or must be addressed outside the network.

Unauthorized subcontracting. Customers may care about how providers handle packets. For instance, a customer who cares about route diversity or privacy way wish to prevent one ISP from transparently outsourcing transit to another. All IGLOO can enforce is that the ISP authenticated and approved the packet; it cannot ensure the ISP's failure independence, nor prevent the ISP from disclosing the com-

munication to others. If the ISP is honest, the use of vnodes to segregate traffic that may and may not be outsourced could help avoid accidental non-compliance. If the ISP is not, then such considerations will have to be verified and enforced outside the scope of the network architecture.

Unsatisfactory service. A provider may sell access to a vnode with a particular service level agreement then fail to uphold that agreement. In the extreme, the provider could drop all packets, even for communications with valid PoCs. In this regard, IGLOO is no worse than the status quo. However, it would be nice for a network architecture to provide some accountability framework to help place the blame on entities violating their SLAs. One concern is that IGLOO could hamper efforts at fault localization such as [9, 29].

Replay attacks. An attacker who has observed a valid packet can attempt to flood a suffix of the packet's path with copies of the packet. A modest-sized replay cache will defeat obvious, aggressive flooding with copies of a small number of packets (§3.5). An attacker who can amass packets from many flows in a single PoC validity window, however, may be able to defeat a replay cache. Should such attacks prove realistic, devising appropriate defenses will be future work.

Unwanted traffic. To avoid carrying troublesome data plane traffic, a realm must ensure that the objectionable path is not authorized. For this, the realm can simply neglect to issue a PoC for the path. But what if the realm issues a PoC and then regrets it? In this case, the realm must either wait for the PoC to expire or immediately invalidate it by changing the vnode key for the local vnode. Note that rekeying may cause a noticeable pause in valid communications, so is primarily appropriate in emergencies.

Of course, an entity may care about much more than dropping the traffic: it may wish to avoid *receiving* it in the first place. In fact, unwanted traffic should ideally be dropped as early as possible, to conserve all participants' network resources. Such early dropping requires that honest realms issue PoCs only when they can verify that all other realms approve. Then unwanted traffic is dropped at the first honest realm. In the next section, we describe a mechanism for such mutual verification in the control plane, but first we ask how to protect the control plane itself.

Control plane flooding. What if, in analogy with "denial-of-capability" [7, 67], attackers target *control plane* servers, which could be easily accessible (e.g., via a public vnode)? This attack points to a question more general than the control plane: how, under IGLOO, can an entity defend against a flooding sender that holds the permissions to reach it?

If clients can be identified at a useful granularity (e.g., "employees", "paying customers", "unknown clients who solved a CAPTCHA"), then the victim can assign each category to a different vnode. When overloaded, the victim deprioritizes categories by not renewing expired PoCs for their vnodes; downgrading service to them; or, in an emergency, changing vnode keys. An organization may also disclose a vnode key to its employees. If the organization's providers fair queue by vnode, then employees are guaranteed to be able to reach their employer, even in the face of massive packet floods. If clients cannot be assigned to categories, we (blatantly) borrow a mechanism from TVA [67]: a victimized realm or its providers can apply Hierarchical Fair Queuing based on the packet's path, to ensure roughly fair bandwidth consumption among senders. Note that while an attacker can weaken this defense under TVA by faking path identifiers, IGLOO's properties prevent this weakening.

5 Control plane flexibility

To explore IGLOO's generality, this section describes how lighting up different controls (who exercises approve/deny? based on what?) can cause IGLOO's data plane to enforce the policy considerations of different policy proposals. We consider BGP, Pathlets [28], TVA [67], NIRA [65], and a new proposal, *sink routing*, and show how routes are disseminated and controls enabled via data plane primitives. In actually enforcing the policies expressed by the participants, IGLOO often provides a bonus relative to the original. For example, today, a BGP advertisement is just... an advertisement: nothing constrains packet flow to obey policy [48].

The section also introduces control plane components and concepts—path servers, public vnodes, consent certificates, and consent servers—that would be more generally useful for other possible IGLOO control planes.

For lack of space, this section is highly abridged; our technical report [51] has more detail. One nagging question that we do not have space to fully answer is how, if communication requires approval, one gets approval to request approval. One (but not the only) answer is that entities can delegate (by publicly disclosing) particular vnodes, the result being that all network entities de facto have permission to use those vnodes. We explain in our technical report how this process is bootstrapped when an end-host attaches to a network. For now, though, it may be useful to regard control plane traffic as traveling over a separate network infrastructure, like the legacy Internet—even though in both our design and implementation, the control traffic travels over IGLOO itself.

BGP. IGLOO permits a path dissemination and path retrieval protocol that chooses the same paths through the network that BGP would. At a high level, realms run BGP between themselves, distributing the equivalent of the advertisements that they would today, except using a flat namespace with signed messages. This approach to route dissemination is shown to be feasible by the authors of AIP [5], and needs small modifications in our environment. First, providers cannot filter routes to customers because the concept of default route does not exist in this design: every realm must know how to reach every destination. Second, advertisements come with a required vnode and vnode key, which allows any entity to mint a PoC to use the advertised realm as a hop but allows the realm to enforce local transit policies [28], such as valley-free routing, as explained in §3.2. Because anyone on the network can mint a PoC for this

vnode, we call it a *public vnode*.

How do senders determine the path to a destination? End-hosts contact a local *path server*. When a sender makes a request to its local path server, the server uses the state collected from BGP advertisements to identify the AS path and to mint the PoCs needed to use the path. The state required in path servers is only a linear factor more than what AIP [5] requires, but AIP requires this state in *routers*, so commodity servers should have no trouble.

TVA [67] and default-off. IGLOO supports a control plane in which, for a sender to send packets to a receiver, it needs authorization from the receiver. If a sender lacks this authorization, the intermediate realms drop its packets. At a high level, this policy—receiver exercises approve/deny based on sender, intermediates apply approve/deny based on receiver's preferences—is what capabilities and other default-off proposals [11, 24, 34, 63, 67] provide. As above, realms run BGP among themselves. Here, advertisements contain two vnodes: a public and a private. The sender gets a path (from its path server) and sends a message along the public vnodes requesting the receiver's permission. If the receiver approves, it signs[3] a statement agreeing to the private vnode path. This statement is called a *consent certificate* and proves the receiver's authorization to the other participants. The receiver sends the consent certificate back along the path. In each realm, a *consent server* checks the consent certificate, generates a PoC for the private vnode path, appends this PoC to a log, and forwards back to the sender. This process allows the sender to reach the receiver on private vnodes. Effectively, the public vnodes implement TVA's capability request channel; the private vnodes, its data plane.

Sink routing. A slight modification to the above enables what we call *sink routing*, where the receiver picks a packet's path. When a capability request reaches the receiver, it uses its path server to pick a (possibly new) path for the sender to use. The receiver then signs a consent certificate for this path, and sends it back through the consent servers, as above.

NIRA [65]. We have built a control plane similar to NIRA [65], where a valid path adheres to the policies of both senders and receivers and is valley-free. We begin by describing the routing protocol *sIRP* .

sIRP (simple IGLOO Routing Protocol) is a link-state protocol that propagates consent certificates as link state advertisements, which declare that the issuing realm is willing to carry traffic between two neighbors, perhaps on particular vnodes. The exact format, together with proposed extensions, is in [51]. A provider R sends its customers the consent certificates that it received along with one expressing that the customer can transit R's network. As a result, each edge customer gets a set of consent certificates that validate paths to well-connected providers (e.g., the Internet core) and to intermediate realms (e.g., its provider's peers). These certificates are installed in path servers.

Path servers arrange themselves in a hierarchy similar to DNS. To reach a destination, a sender queries its local path server, which then takes over. That path server, X, contacts another path server in the hierarchy, Y, supplying sets of consent certificates that prove the sender has permission to reach well-connected intermediate realms. Y looks for an intermediate realm that the sender can reach, and that the receiver can hear from, and then constructs a full path (as in NIRA). If Y does not know how to reach the destination, it responds with a path and consent certificates to reach the next path server in the hierarchy. The required number of round-trips is equivalent to DNS (assuming, loosely speaking, that path servers know paths to subordinates and superiors in the hierarchy), and the result is that the sender gets a path to the destination. But where do the needed PoCs come from? The answer is that PoC issuance has been delegated to the path servers, so the messages above actually contain the needed PoCs as well. The above process, though tedious to read about, boils down to this: paths are constructed based on senders' and receivers' preferences, but they obey the policies of the intermediate realms, with PoC issuance delegated from the core to the edges (that is, to the sender's local path server, and to the receiver's path server).

We now briefly describe one extension to sIRP consent certificates that allows additional controls to be enabled.

Pathlets. Using a similar approach to the one above, IGLOO can provide the equivalent of Pathlet routing [28]. Consent certificates are extended to specify a pathlet (this extension is described in [51]). Indeed, IGLOO borrowed vnodes from Pathlet routing in the first place.

6 Implementation

This section describes our implementation of the hardware and software data plane, the control plane, and endpoint software. All of our software runs on Linux 2.6.25.

Data plane. Our prototype forwarder accepts IGLOO packets carried in Ethernet frames and implements the protocol described in §3.6. The fast path runs on the NetFPGA programmable hardware platform [2], which has 4 GigE ports. When an IGLOO packet enters the fast path, if the packet's path contains one or more realms R_j for which the forwarder, representing realm R_i, does not have $k_{i,j}$ cached in hardware, the hardware sends the packet to a software slow path over the PCI bus to the processor. The slow path, implemented in Click [40], calculates the needed keys and installs them in the hardware's key cache. The Diffie-Hellman key exchange is implemented with the MIRACL cryptographic library [57]. We have not yet implemented PoC expiry via the expiration field, handling error packets, or replay prevention. Adding PoC expiry or error handling should not change our evaluation significantly in the next section, nor should simple replay prevention.

The hardware image uses support modules from the NetFPGA project. We implemented the IGLOO-specific

[3] Realms also use their private keys to generate the $k_{i,j}$ (§3.2). Such "dual purposing" of key material is wisely discouraged by folklore. However, an analysis, which is outside of our scope, indicates that our protocols are safe.

Machine type	CPU	RAM	OS
slow	Intel Core 2 Duo 1.86 GHz	2 GB	Linux 2.6.25
medium	Intel Core 2 Quad 2.40 GHz	4 GB	Linux 2.6.25
fast	Intel quad Xeon 3.0 GHz	2 GB	Linux 2.6.18

Table 3—Machines for measuring IGLOO overhead.

Varied parameter	Range	Fixed parameters		
		Pkt size	Path len	Path idx
Packet size	{311, 567, 823, 1335, 1514}	—	7	3
Path length	{3, 7, 10, 20, 30, 35}	1514	—	1
Path index	{1, 5, 10, 15, 18}	831	20	—

Table 4—Parameters used throughout experiments. Packet size includes header.

logic, including cryptographic modules. The forwarder has a total equivalent gate count of 13.4M and uses 89% of the total FPGA logic area. By comparison, the NetFPGA reference IP router has an equivalent gate count of 8.7M and uses 50% of the total FPGA logic area.

Control plane and endpoints. Our combined consent and path server is embedded in a DNS-like naming hierarchy and exposes a `getpath()` call over XDR RPC (which returns a path to a destination or to another such server). These servers participate in sIRP. The control plane modules are 1500 semicolons of C++, not including cryptographic libraries. To send an IGLOO packet, an endpoint application calls `getpath()`. This function returns an opaque handle that the application uses as the destination IP address, which a local Click instance then translates into an IGLOO header.

7 Evaluation

IGLOO introduces space and time overhead from per-packet cryptographic objects and operations. Our principal question in this section is whether these overheads are practical on Internet backbone links. We begin by estimating IGLOO's total space overhead (§7.1). In §7.2 and §7.3 we present microbenchmarks that evaluate the performance of our prototype forwarder and the supporting software, respectively. In §7.4, we extrapolate from our results to assess IGLOO's future feasibility in the Internet core. Finally, in §7.5, we consider the overhead of IGLOO's control plane.

Setup and parameters Table 3 lists the three machines that we used to evaluate IGLOO. Different experiments use different machine classes to simulate real usage. The NetFPGA used in the experiments is in the *slow* machine.

Our experiments often vary packets' path lengths, path indices or sizes. Table 4 gives the fixed and variable parameters used for the forwarding throughput and software performance measurements.

7.1 Packet overhead

Relative to IP, IGLOO requires larger packet headers; here we elaborate on its projected effect on bandwidth consumption. Because we cannot predict future traffic patterns (to do so requires a separate study), our analysis uses current estimates of AS path lengths and current estimates of packet sizes.

The fields in an IGLOO header that do not depend on the packet's path length use 13 bytes (see Figure 1). For each realm in the path, the hop info requires 42 bytes: 24 bytes for the hop ID (R_i, r_i), and 18 bytes for the hop verifier V (see §3.6). For a packet whose path length is 7—the average AS-level path length in [37] but a high estimate, according to [5]—the header is 307 bytes, or 20.3% of a 1514-byte packet. For small packets, IGLOO's overhead would be far larger, but note that most bytes travel in large packets: the average packet size reported by [59] is 1370 bytes.

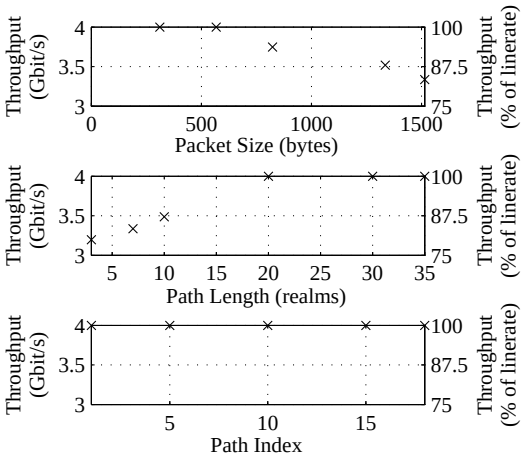


Figure 2—Avg throughput as a function of packet size (Table 4 row 1), path length (Table 4 row 2), and path index (Table 4 row 3). Percentages relative to max throughput possible on NetFPGA. Standard deviation is less than 0.02% of the mean at each measurement point. The forwarder's throughput is lowest for packets with large payloads but small path lengths, when the most bits must be hashed.

Thus, on average, IGLOO would add overhead of $307/1370$, meaning that IGLOO would increase *total* bandwidth consumption by 22.4% relative to IP, assuming today's packet size distributions. This amount is sizable, but it may be a fair price for IGLOO's properties, as discussed in §8.

7.2 IGLOO forwarder

We now measure the throughput of our prototype forwarder.

From §3.6, one might expect the cost of processing a packet to depend on the path length because the work of "verifying" and "proving" seems proportional to the path length. However, the results of the various PRF-96 operations are XORed, so they can be parallelized and thus removed from the critical path. The only other heavily serialized function in the design is the hash function (H-247), so we expect it to be the bottleneck; i.e., throughput should depend on the number of bits that must be hashed. Since the only fields that are *not* hashed are the hop # and the verifiers V_i, we expect throughput to be lower when the V_i represent a smaller fraction of the total packet bits. In other words, for a constant path length, we expect throughput to decrease as packet size increases. And for a constant packet size, we expect the throughput to increase as the path length increases.

To validate, we measure our prototype's fast path throughput by connecting the four ports of an IGLOO forwarder to a

Action	Processing time	Throughput (1/Proc. time)
Calculate $k_{i,j}$	4 ms ($\sigma = .043$ ms)	250 keys/s
Calculate path	87 μs	11505.6 path/s
Generate PoC	$0.4l + 1.3$ μs	$2.6 \cdot 10^6/(l + 3.5)$ PoC/s
Create packet (w/cache)	$2.6l + 40.1$ μs	$3.9 \cdot 10^5/(l + 15.4)$ pkt/s
Verify packet (w/cache)	$2.6l + 24.4$ μs	$3.9 \cdot 10^5/(l + 9.5)$ pkt/s
Create packet (no cache)	$33796.1l - 32758.4$ μs	$29.6/(l - 0.9)$ pkt/s
Verify packet (no cache)	$34875.1l - 33647.1$ μs	$28.6/(l - 0.9)$ pkt/s

Table 5—Processing time and throughput for software operations, where l is the path length. Packet creation and verification costs are measured both with and without the use of cached shared keys. For the last five rows, processing time is derived by linear regression, and $R^2 > 0.99$ in all three cases.

NetFPGA packet generator that sends IGLOO packets at line rate. We measure throughput over 5 10-second samples, using the measurement points in Table 4. The IGLOO forwarder loops ingress packets back to the packet generator, which measures the average bit rate.

Figure 2 plots the measured throughput. Note that we do not report goodput; instead we acknowledge that IGLOO has a 22.4% overhead, as analyzed in §7.1. The minimum aggregate throughput is 3.3 Gbit/s. By comparison, IP itself only runs at 4 Gbit/s on this hardware platform. The path index has no effect on performance because it doesn't affect the number of PRF-96 applications or the number of bits hashed.

7.3 Data plane software performance

We now measure the performance of IGLOO's data plane software. We focus on the forwarder's slow path, the main cost of which is calculating shared keys (§3.4), and end-host packet operations. Table 5 summarizes.

Shared key ($k_{i,j}$) calculation. We measure the cost of the shared key derivation by running 3000 iterations of the calculation function in a tight loop on the *slow* machine. On average, a single calculation takes 4 ms. However, a shared key cache for all realms can fit into SRAM (see §7.4), so on a real forwarder, this calculation would take place rarely.

End-host. An end-host must also perform cryptographic operations: senders initialize all hop verifier entries, and receivers validate and modify some of these entries. To understand these costs, we seek a linear function from path length to processing time. To infer such a function, we vary path length per Table 4, take packet size to be 1514 bytes, and collect 1000 samples per path length on the *medium* machine. We record total processing cost (of either packet generation or verification, depending on sender or receiver; in both cases, we record the cost when the $k_{i,j}$ keys are and are not cached), and then use ordinary least squares linear regression. The inferred coefficients ($R^2 > 0.99$) are in Table 5. Each entry in the path increases packet creation and verification times by 2.6 μs, the cost of two AES encryptions. For an average path length of 7 (from [37]), packet verification can be performed at 21K pkt/s.

Note that an end-host takes longer to generate a packet than to verify one. This is because senders are so far unoptimized and compute $H(P, M)$ twice. Were the endpoints optimized, receiving would likely be more expensive than sending: the receiver also hashes the packet (to verify V) and has an additional cost, namely re-computing the local PoC.

	NetFPGA IGLOO	NetFPGA IP	Commercial IP (est.)
Min Throughput (Gbits/s)	3.3 (from §7.2)	4	1
(Eq.) Gate Count (Gates)	13.4M	8.7M	5M
Normalized Cost (Gates/(Gbits/s))	4.1M	2.2M	5M

Table 6—Normalized costs of the NetFPGA IGLOO forwarder, the NetFPGA IP router, and an estimate of a commercial IP router.

7.4 Scaling

Here, we present an extremely brief assessment of whether a production implementation of IGLOO could meet the demands of the Internet backbone. We elaborate in [51].

Throughput. To assess whether IGLOO could scale to backbone speeds, we compare IGLOO to IP in terms of *normalized cost*, which measures the hardware cost, reported as equivalent gate count, per unit of throughput. We consider two IP implementations: the NetFPGA reference IP router and an informal estimate of a commercial IP router. We obtain gate counts for the NetFPGA implementations from the synthesis produced by Xilinux's ISE software, and we estimate the cost of an enterprise-grade commercial IP router's forwarding engine and switch fabric (i.e., excluding queues and management hardware) at around 5M gates per port.

Table 6 summarizes the comparison. The NetFPGA IGLOO forwarder is ∼86% more expensive than the NetFPGA IP router—but a little cheaper than the commercial router. Since commercial IP routers can scale to backbone speeds (around 100 Gbits/s) and since almost all of IGLOO's processing can be parallelized, it seems that there is no fundamental obstacle to scaling IGLOO to such speeds.

Symmetric key cache. An IGLOO forwarder stores a table of (R_i, $k_{i,j}$) pairs. Even under the (conservative) assumption that a forwarder caches a $k_{i,j}$ for every other realm in the Internet, the memory requirements are not likely to be onerous: today's routers already hold on the order of hundreds of thousands of prefixes in hardware for forwarding [20]. Meanwhile, there are fewer than 33k AS numbers, and the total is growing at less than 3.2k/year [1]. Under a rough equivalence between realms and ASes, the symmetric key cache would thus fit easily into a current IP router's memory. For further analysis of a nearly identical question, see [5, §4].

Vnode key cache. An IGLOO forwarder also caches precalculated vnode keys or prefix keys (§3.6). While an extended analysis is outside of our scope, we just note the following two use cases. Viable SRAMs already exist that could fit 2^{20} vnode keys, sufficient to give 1 million customers a vnode each. The cache could also be used to store prefix keys. So if delegation occurs at, say, 12-bit boundaries, the forwarder can cache the keys for all of its 20-bit prefixes, deriving the needed vnode key per-packet. Each derivation requires one AES invocation so would not affect throughput.

7.5 Control plane overhead

Here, we explore the computation costs associated with the IGLOO control plane that is based on NIRA, described in §5. The results are summarized in Table 5.

Path negotiation. To measure the cost of building paths, we run the path building function in a tight loop on a *fast* machine. We configure the server with paths to 100 destinations. Our results show that our prototype path server's throughput is approximately 11K paths/s. While this is an order of magnitude fewer than the number of DNS requests a fast server could handle, our current implementation is unoptimized.

PoC retrieval. To measure the cost of generating a PoC, we run the calculation function in a tight loop, varying the path length per Table 4. To represent the hardware that runs IGLOO control plane servers, we use the *fast* machine. Our results show that the cost is proportional to the path length, as expected from the definition of PoC.*proof* (§3.3).

8 Discussion

Expressiveness. While we obviously cannot prove that IGLOO's controls are sufficient, they appear, as mentioned earlier, to encompass those of prior transit policy proposals. By making the inter-domain path available to entities in their approval decisions, IGLOO permits the expression of the works in Table 1. By allowing entities to associate an opaque tag (the vnode) to a path, IGLOO permits a slew of local resource policies. Some of these were enumerated in §3.1–§3.3. Others are as follows: an IGLOO entity can force a packet through an internal middlebox by approving only flows that transit vnode *v*, where *v* internally maps to an intra-domain route that travels through the middlebox. An IGLOO entity can use the vnode to specify policies about time of day, rekeying the vnode as the time changes. Etc. And by allowing entities' authorization decisions to take arbitrary input prior to packet flow, IGLOO permits an entity to express arbitrary policies, including that the sender has paid its bill, that the user is authenticated, etc.

Note that an ideal definition of enforcement has the network dropping packets at the source, not at the disapproving entity. IGLOO upholds this notion but requires control plane coordination for it (§5). To do so without such coordination (e.g., a packet contains signed statements from each realm specifying its policies, allowing the first hop to drop if the last hop did not approve) seemed infeasible at high speeds.

Non-goals and limitations. We note that while IGLOO does enable many new functions, there are some it is not designed to accommodate. As mentioned in §2 and §4, IGLOO is not designed to prevent realms from transparently delegating or collaborating with each other. What it can do is bring the various policy decisions into the open, so that, for instance, a sender can at least *choose* which providers to trust.

IGLOO's limitations derive from its reliance on source routing: entities cannot hide topology or route information from those who carry and receive traffic, forward packets differently based on their payloads (unlike [54]), or modify a packet's payload during transit.

Overhead. As described in §7, IGLOO introduces logic area cost and packet space overhead. These may seem high, but we must weigh them against the benefits of IGLOO's additional properties. We are not in a position to assess this trade-off today, but we just note that technology trends ought to continue to reduce these costs (under jumbo frames, IGLOO's packet space overhead would be trivial). Indeed, one way to look at IGLOO is that it spends some of our ever-increasing bandwidth and processing resources to get properties currently unavailable today.

Deployment. IGLOO can be deployed incrementally in a manner similar to Platypus [55]: IGLOO forwarders can treat IP as the link layer, rewriting IP source and destination addresses at each IGLOO hop. Realms that do deploy IGLOO gain incremental benefits: IGLOO-enabled providers are able to establish identities of IGLOO-enabled senders (for accounting purposes, for example), IGLOO-enabled senders can select different levels of service from IGLOO-enabled providers by choosing vnodes, and receivers whose policy requires it can check that traffic has passed through an IGLOO-enabled virus-scanning service.

9 Related work

IGLOO borrows much from many. Its data plane mechanisms individually resemble prior mechanisms: realm IDs are like AIP's ADs [5]; PoCs are similar to capabilities [55, 67] and Visas [26]; vnodes were introduced in Pathlet routing [28]; and PoPs rely on a construction like the one in [8]. IGLOO's control/data split was inspired by [15, 17, 30, 31], and sIRP was inspired by NIRA [65]. The novelty of IGLOO is in its overall architecture, which composes these and our own mechanisms into a coherent design. In doing so, IGLOO expresses a larger set of policy considerations than prior work, and enforces a stronger set of properties. We attempt to make this point in Tables 1 and 2, which contain lists of proposals that motivated our work.

10 Summary

> Solutions that are less efficient from a technical perspective may do a better job of isolating the collateral damage of tussle [22].

We began by asking what a general policy framework for the future Internet might look like: what policies should be supported, and can they be enforced? We then proposed a policy principle: any entity along the path of a communication, including the endpoints, gets to approve or deny communications. The entity may take into account (1) the inter-domain path; (2) what internal resources it would allocate to the communication; and (3) arbitrary factors. These arbitrary factors could include other realms' preferences (but not their internal packet handling). None of this was to say that all such controls *would* be "lit up", only that we cannot predict which controls will be most important, so we should permit their simultaneous expression. The tussle between vari-

ous entities (e.g., end-hosts vs. providers) can then take place within the context of the design, not by violating it [22].

This paper makes contributions to policy and mechanism. For policy, we observe that (1)–(3) are a superset of prior policy proposals. On the mechanism side, we showed how a network architecture, IGLOO, could enforce this very large set of potential policies with a very small number of mechanisms (realm ids, vnodes, PoCs, PoPs, and the overall control/data split) while obeying constraints of the Internet environment like tolerating adversarial behavior, the need for efficient forwarding hardware, and the absence of centralized authority. Along the way, we solved a longstanding problem in network architecture: how to bind a packet to its path in a federated environment. And to do that, we introduced packet authentication techniques that may be of independent interest. We validated our design by implementing it on the NetFPGA platform and observing that, though it is not cost-free—it asks more of forwarders, and it requires larger packet sizes—it is within the realm of plausibility.

Certainly, IGLOO is not perfect, but we think it broadens the space of the possible. In particular, it responds to two formerly unanswered questions in network architecture: how can we uphold the many policy considerations *simultaneously*, and what do we have to pay to do so?

References

[1] The 32-bit autonomous system number report. `http://www.potaroo.net/tools/asn32/index.html`.

[2] NetFPGA: Programmable networking hardware. `http://netfpga.org`.

[3] The OpenFlow switch specification. `http://OpenFlowSwitch.org`.

[4] Digital signature standard (DSS). Federal Information Processing Standards Publication, November 2008. DRAFT FIPS PUB 186-3.

[5] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet protocol. In *SIGCOMM*, Aug. 2008.

[6] K. Argyraki and D. R. Cheriton. Loose source routing as a mechanism for traffic policies. In *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, Sept. 2004.

[7] K. Argyraki and D. R. Cheriton. Network capabilities: The good, the bad and the ugly. In *HotNets*, Nov. 2005.

[8] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *INFOCOM*, Mar. 2004.

[9] I. Avramopoulos and J. Rexford. Efficient data-plane security for IP routing. In *USENIX Technical Conference*, June 2006.

[10] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. RFC 3209, Dec. 2001.

[11] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! In *HotNets*, Nov. 2005.

[12] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Proc. EUROCRYPT*, pages 384–397, 2002.

[13] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. RFC 1633, June 1994.

[14] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, Sept. 1997.

[15] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *NSDI*, May 2005.

[16] K. Calvert, J. Griffioen, and L. Poutievski. Separating routing and forwarding: A clean-slate network layer design. In *Proc. IEEE Broadnets*, Sept. 2007.

[17] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM*, Aug. 2007.

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

[19] J. Chou, B. Lin, S. Sen, and O. Spatscheck. Proactive surge protection: a defense mechanism for bandwidth-based attacks. In *USENIX SECURITY*, July 2008.

[20] Cisco Systems, Inc. Cisco Catalyst 6500/Cisco 7600 Series Supervisor Engine 720 Datasheet. `http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/product_data_sheet09186a0080159856.pdf`.

[21] D. Clark. Policy routing in internet protocols. RFC 1102, May 1989.

[22] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. In *SIGCOMM*, Aug. 2002.

[23] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.

[24] C. Dixon, T. Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *NSDI*, Apr. 2008.

[25] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala. Source demand routing: Packet format and forwarding specification (version 1). RFC 1940, May 1996.

[26] D. Estrin, J. Mogul, and G. Tsudik. VISA protocols for controlling inter-organizational datagram flow. *IEEE JSAC*, 7(4), May 1989.

[27] D. Estrin and G. Tsudik. Security issues in policy routing. In *Proc. IEEE Symposium on Security and Privacy*, May 1989.

[28] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *SIGCOMM*, Aug. 2009.

[29] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. In *SIGMETRICS*, June 2008.

[30] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM CCR*, 35(5), Oct. 2005.

[31] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM CCR*, 38(3):105–110, July 2008.

[32] S. Guha and P. Francis. An end-middle-end approach to connection establishment. In *SIGCOMM*, Aug. 2007.

[33] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *OSDI*, Dec. 2004.

[34] M. Handley and A. Greenhalgh. Steps towards a DoS-resistant Internet architecture. In *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, Aug. 2004.

[35] P. Hawkes and C. McDonald. Submission to the SHA-3 competition: The CHI family of cryptographic hash algorithms. Submission to NIST, 2008. `http://ehash.iaik.tugraz.at/uploads/2/2c/Chi_submission.pdf`.

[36] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *NDSS*, 2002.

[37] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a Tier-1 IP backbone. In *INFOCOM*, 2003.

[38] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Topics in Cryptology – CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 155–169, April 2008.

[39] H. T. Kaur, A. Weiss, S. Kanwar, S. Kalyanaraman, and A. Gandhi. BANANAS: An evolutionary framework for explicit and multipath routing in the internet. In *Proc. SIGCOMM Workshop on Future Directions in Network Architecture*, Aug. 2004.

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

[41] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM*, Aug. 2007.

[42] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *ACM/IEEE Trans. on Networking*, 9(3):293–306, June 2001.

[43] M. Little. Goals and functional requirements for inter-autonomous system routing. RFC 1126, Oct. 1989.

[44] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and adoptable source authentication. In *NSDI*, Apr. 2008.

[45] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *SIGCOMM*, Aug. 2008.

[46] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM CCR*, 32(3), July 2002.

[47] R. Mahajan, D. Wetherall, and T. Anderson. Mutually controlled routing with independent ISPs. In *NSDI*, Apr. 2007.

[48] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an accurate AS-level traceroute tool. In *SIGCOMM*, Aug. 2003.

[49] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *SOSP*, Dec. 1999.

[50] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *IEEE DSN*, June 2005.

[51] J. Naous, A. Seehra, M. Walfish, D. Mazières, A. Nicolosi, and S. Shenker. The design and implementation of a policy framework for the future Internet, Sept. 2009. `http://www.cs.utexas.edu/~mwalfish/icing-tr-09-28.pdf`.

[52] R. Perlman. *Network layer protocols with Byzantine robustness*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[53] R. Perlman. Routing with Byzantine robustness. Technical Report TR-2005-146, Sun Microsystems, Aug. 2005.

[54] L. Popa, I. Stoica, and S. Ratnasamy. Rule-based forwarding (RBF): improving the Internet's flexibility and security. In *HotNets*, Oct. 2009.

[55] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *SIGCOMM*, Sept. 2004.

[56] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching. RFC

3031, Network Working Group, Jan. 2001.
[57] M. Scott. Miracl library. https://www.shamus.ie/index.php?page=Downloads.
[58] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, Aug. 2002.
[59] The Cooperative Association for Internet Data Analysis (CAIDA). Packet size distribution comparison between internet links in 1998 and 2008. http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml.
[60] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI*, Dec. 2004.
[61] G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *INFOCOM*, Mar. 2005.
[62] W. Xu and J. Rexford. MIRO: Multi-path interdomain routing. In *SIGCOMM*, Sept. 2006.
[63] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *Proc. IEEE Symposium on Security and Privacy*, May 2004.
[64] A. Yaar, A. Perrig, and D. Song. StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense. *IEEE JSAC*, 24(10):1853–1863, Oct. 2006.
[65] X. Yang, D. Clark, and A. W. Berger. NIRA: A new inter-domain routing architecture. *ACM/IEEE Trans. on Networking*, 15(4), Aug. 2007.
[66] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, Sept. 2006.
[67] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM*, Aug. 2005.
[68] X. Zhang, A. Jain, and A. Perrig. Packet-dropping adversary identification for data plane security. Dec. 2008.